AD-770 632

ANALYTIC MODELS OF TIME-SHARED COMPUTING SYSTEMS: NEW RESULTS, VALIDATIONS, AND USES

John W. McCredie

Carnegie-Mellon University

Prepared for:

Air Force Office of Scientific Research
Defense Advanced Research Projects Agency

November 1972

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

BEST
AVAILABLE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR - TR - 73 - 2075
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
ANALYTIC MODELS OF TIME-SHARED COMPUTING SYSTEMS: NEW RESULTS, VALIDATIONS, AND USES

5. TYPE OF REPORT & PERIOD COVERED
Interim

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
John W. McCredie

8. CONTRACT OR GRANT NUMBER(s)
F44620-70-C-0107

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pennsylvania 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
61101D
AO 827

11. CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, Virginia 22209

12. REPORT DATE
November 1972

13. NUMBER OF PAGES
189

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Air Force Office of Scientific Research (NM)
1400 Wilson Blvd
Arlington, Virginia 22209

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Reproduced by
NATIONAL TECHNICAL INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The goals of the research discussed in this report are: (1) to create new models of time-shared computer systems which include important features commonly found in real systems; (2) to insure that the formulations of, and solutions to, these models are relatively simple so that they may be used by designers and computer center managers; (3) to compare the behavior of these models with the behavior of more complex systems through simulation studies and empirical performance investigations of operational computers; and

192

AD 770 632

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20. Abstract (Continued)

(4) to indicate some of the ways these models may be used to aid in the design, evaluation, and control of time-shared computers. Analytic models are but one of many tools available to those who want to analyze, measure, improve, and create better computing systems. One of the goals of this report is to help place this approach to system modeling into perspective as an important tool, not a panacea, for computer scientists.

ANALYTIC MODELS

OF TIME-SHARED COMPUTING SYSTEMS:

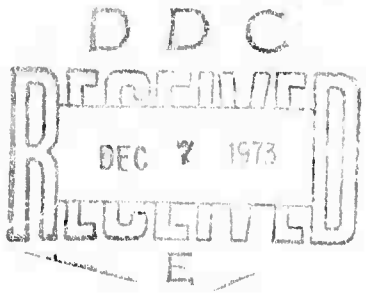
NEW RESULTS, VALIDATIONS, AND USES

by

John W. McCredie

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

November, 1972



Submitted to Carnegie-Mellon University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This work was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under contract (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release; its distribution is unlimited.

16

ACKNOWLEDGMENTS

I would like to thank all of the people who create, at Carnegie-Mellon University, a stimulating environment in which one is encouraged to undertake interdisciplinary studies. My special thanks go to Professors Kriebel, Lehoczky, and Newell for their many valuable suggestions. Dorothy Josephson is able to make equations, tables, and figures look like works of art, and I want to thank her for her preparation of the manuscript.

This work is dedicated to my wife and two children without whose help it might have been completed two years sooner, or maybe never.

ABSTRACT

The goals of the research discussed in this report are:

(1) to create new models of time-shared computer systems which include important features commonly found in real systems;

(2) to insure that the formulations of, and solutions to, these models are relatively simple so that they may be used by designers and computer center managers;

(3) to compare the behavior of these models with the behavior of more complex systems through simulation studies and empirical performance investigations of operational computers; and

(4) to indicate some of the ways these models may be used to aid in the design, evaluation, and control of time-shared computers.

Chapter 1 contains an introduction to some important features of current time-shared computers and a survey and review of many of the current approaches to their modeling. Errors in three well known articles are discussed and corrected.

Chapter 2 presents a number of new models which are extensions to, and modifications of, previous studies. The new features include a more realistic treatment of overhead degradation and processing quantum length. One of the models is a feedback queueing structure having two servers in tandem. The results of each model include the mean value of the time required by the system

to respond to a request. In addition, exact and approximate expressions for expected response time conditioned on service request are developed and compared with each other to study the accuracy of the approximations.

Chapter 3 presents the results of a number of simulation experiments designed to examine the robustness of the analytic models. The first model is similar to the first analytic formulation. The next two simulations are based on the tandem queueing structure. The last simulation includes a detailed model of the scheduling mechanism of TSS/360, an operational time-shared system marketed by IBM.

Chapter 4 contains the results of three empirical studies of actual systems. The first two were performed on TSS/360 and the third was performed on a Univac 1108 running with EXEC-8, a time-sharing operating system.

Chapter 5 contains a discussion of some applications of the models developed in Chapter 2. The first example is an application of the models to a design decision for the operating system of a multi-processor configuration. The next illustrates the way the models may be used in performance evaluation studies to examine possible overall system improvements arising from enhancements to subsystems. The last example indicates how the models may be used in a dynamic control system to improve system performance.

The first appendix presents a number of results which were used, but not derived, in earlier chapters. The remaining appendices contain listings of major programs used in the research.

Analytic models are but one of many tools available to those who want to analyze, measure, improve, and create better computing systems. One of the goals of this report is to help place this approach to system modeling into perspective as an important tool, not a panacea, for computer scientists.

TABLE OF CONTENTS

TABLES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 THE USE OF ANALYTIC MODELS - AN OVERVIEW

> "Everyone today knows that a queue is a waiting line. If one also takes the trouble to examine the literature, which now is nearing 2000 references on the subject, he might get the idea that all those contributing to the understanding of congestion phenomena are interested in doing something about them since, after all, queueing theory is concerned with relieving pain and saving time for all of us who have to wait. Indeed, queues make a substantial demand on our very lives by taking precious time from them.
>
> But the situation is getting worse in spite of the fact that in the past seven years the literature of queueing theory has increased by half of its amount for the previous fifty years. Improvements do not match the increase in theoretical developments. Rarely has so much ingenuity been shown in tackling a variety of technical problems on paper by some of the ablest people in the world. It may be that many additional good papers are waiting in queues for publication. But real life queues are still primitive, and indifference to waiting by both facility owners and resigned customers is a normal state of affairs."
>
> Thomas L. Saaty[1]

> "The big problem with management science models is that managers practically never use them. There have been a few applications, of course, but the practice is a pallid picture of the promise. Much of the difficulty lies in implementation and an especially critical aspect of this is the meeting between manager and model. I believe that communication across this interface today is almost nil and that the situation stands as a major impediment to successful use of models by managers .....A model that is to be used by a manager should be simple, robust, easy to control, adaptive, as complete as possible, and easy to communicate with."
>
> John D. C. Little[2]

The goals of the research discussed in this report are: (1) to create new models of time-shared computer systems which include important features commonly found in real systems; (2) to insure that the formulations of, and solutions

---

[1] Saaty, T., "Seven More Years of Queues, A Lament and A Bibliography", Naval Research Logistics Quarterly, Vol. 13, No. 4, December, 1966, p. 447.

[2] Little, John D. C., "Models and Managers: The Concept of a Decision Calculus", Management Science, Vol. 16, No. 8, April 1970, p. B-466.

to,these models are relatively simple so that they may be used by designers and computer center managers; (3) to compare the behavior of these models with the behavior of more complex systems through simulation studies and empirical performance investigations of operational computers; and (4) to indicate some of the ways these models may be used to aid in the design, evaluation, and control of time-shared computers. The quotations from Saaty and Little indicate that often theoretical results of operations research studies are not applied to practical situations. The Institute of Management Science recently changed the name (and the focus) of one of its periodicals to Interfaces in an attempt to bridge this implementation gap.

Two reasons why computing system models often remain unused are that articles describing them seldom contain discussions about their validity for describing observed phenomena and that often the results are so complicated that users are not willing to invest the time needed to understand the model and its behavior. The main purpose of descriptive models is to account for observed phenomena of physical systems. The complexity of most actual systems requires that any particular model address itself to a limited and constrained subset of state variables. Thus each model is an abstraction of a particular set of important features of interest to an analyst or designer. Simplifications required to make an abstraction manageable by a particular solution technique limit both scope and power. Since analytic models are characterized by symbolic formulations and deductive derivations, they require many simplifying assumptions. The consequences of these assumptions must be explored before one applies the model.

For the study of computing systems there are two other tools which are related to analytic modeling:

(1) the construction of large, detailed, simulations

(2) the design and implementation of empirical investigations

All three methods have areas of applicability which intersect. For example, analytic models often expand to the point where a large amount of computational effort is required to calculate results. Often a point is reached when a modest simulation may be a more cost effective approach. Large simulations may eventually grow into system prototypes, and empirical investigations can provide insight required to design better models. Analytic models often indicate which of many possible parameters or subsystems are good candidates for more detailed study via simulation and experimentation. Another important use for analytic models is as a reference system for statistical analysis of simulation results. For example, Gaver (1969) presents evidence showing how the classic Monte Carlo technique of control variates, which makes use of an approximate model, can improve simulation efficiency by reducing the variance of parameter estimates from simulation experiments.

To be useful, analytic formulations should include the essential features of a system, or subsystem, and should have solutions that are readily understandable. The necessity of spending excessive computer effort to solve for each parameter value of an analytic model casts doubt upon its usefulness since simulations typically can handle more detailed cases with similar effort. The conclusion from these considerations is that analytic models, simulation studies, and empirical investigations should complement one another in the study of computing systems. The new models developed in the next chapter add to the tools available for analytic performance analysis.

Section 1.2 contains an introduction to some important features of time-shared computer systems, and Section 1.3 contains a survey and review

of many of the current approaches to their modeling. Errors in three well known articles are discussed and corrected.

Chapter 2 presents a number of new models which are extensions to, and modifications of, previous studies. The new features include a more realistic treatment of overhead degradation and processing quantum length. One of the models is a feedback queueing structure having two servers in tandem. The results of each model include the mean value of the time required by the system to respond to a request. In addition,exact and approximate expressions for expected response time conditioned on service request are developed and compared with each other to study the accuracy of the approximations.

Chapter 3 presents the results of a number of simulation experiments designed to examine the robustness of the analytic models. The first model is similar to the first analytic formulation. The next two simulations are based on the tandem queueing structure. The last simulation includes a detailed model of the scheduling mechanism of TSS/360, an operational time-shared system marketed by IBM.

Chapter 4 contains the results of three empirical studies of actual systems. The first two were performed on TSS/360 and the third was performed on a Univac 1108 running with EXEC-8, a time-sharing operating system.

Chapter 5 contains a discussion of some applications of the models developed in Chapter 2. The first example is an application of the models to a design decision for the operating system of a multi-processor configuration. The next illustrates the way the models may be used in performance evaluation studies to examine possible overall system improvements arising from enhancements to subsystems. The last example indicates how the models may be used in a dynamic control system to improve system performance.

The chapter concludes with an evaluation of the techniques and points to future work.

1.2 IMPORTANT FEATURES OF TIME-SHARED SYSTEMS

> "It is now possible for users to be connected by a pair of wires to a powerful computer system that may be in the next room or may be many miles away. All users, wherever they are, have instant access to the computer, and can expect a response to their demands that is limited only by the fact that the computer must share its time between all the users. The development of such systems is, however, still in its infancy, and much development of hardware and software must take place before users can be given everything that they have a right to demand. There is no doubt that, in a few years time, the best of the currently operating systems will appear very primitive indeed."
>
> M. V. Wilkes[3]

Time-Sharing Computing Systems, by Wilkes (1968), is a good introduction to the hardware and software features included in many time-shared structures. It will provide a good background to the non specialist.

Figure 1.1 illustrates the basic features of many time-shared systems. Users submit tasks from terminal devices to the system. A task may be conceptualized as a job step which requires the use of a number of system resources to be completed. The computer's operating system controls and allocates these resources, such as primary and secondary memory, channels, and processors, so that users requiring small amounts of resources will get a rapid response from the system. In this report response time will be defined as the elapsed time from task submittal to task completion. If a particular task keeps a resource such as the central processor occupied for a time period that would seriously affect the response of other jobs in the system, it is interrupted and placed in the system of queues while another task uses the resource. When the system has finished with all of the work associated with

[3] Wilkes, M. V., Time-Sharing Computer Systems, American Elsevier Publishing Company, Inc., New York, 1968, p. 2.

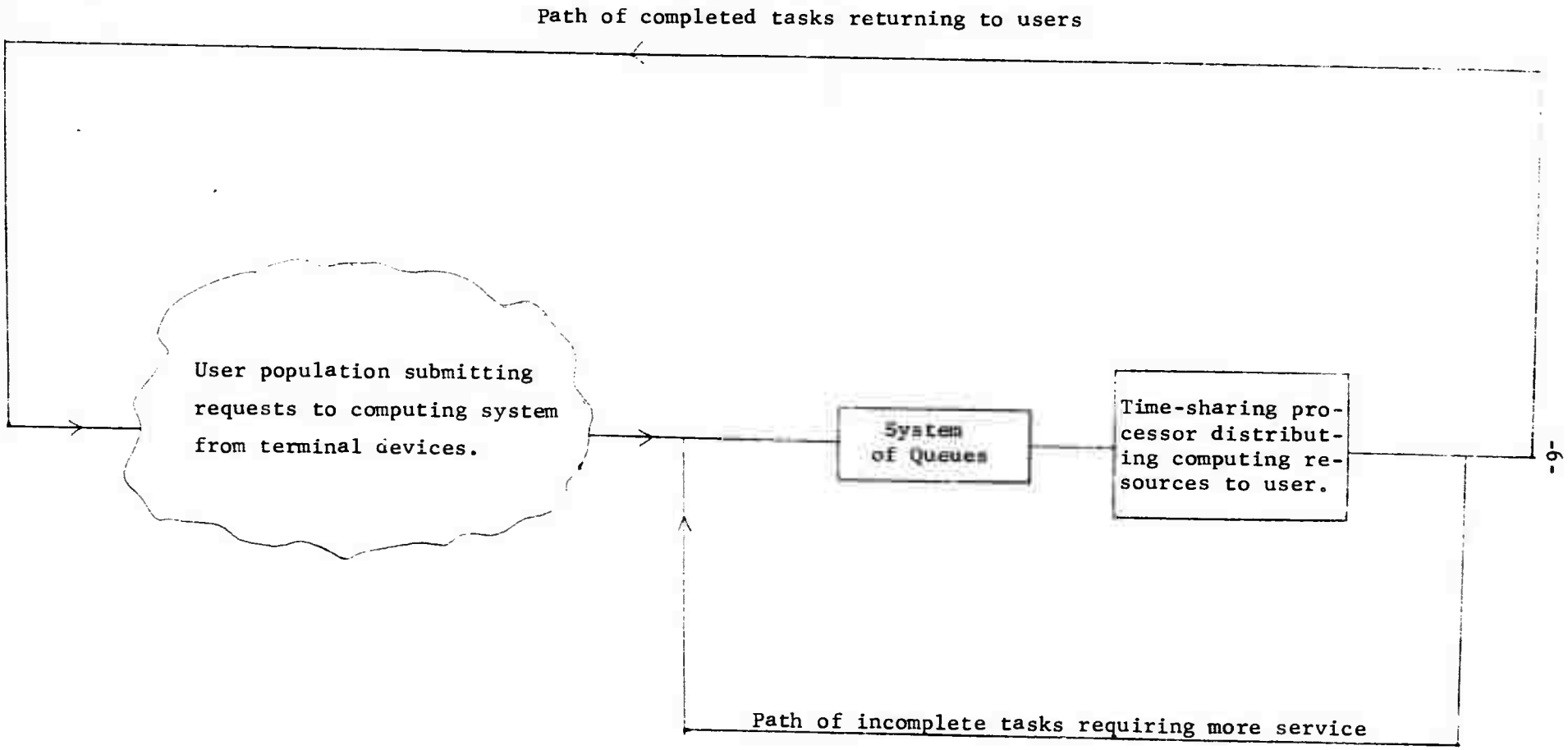


Figure 1.1

General Structure of Time-Sharing Systems

the task it gives the user an appropriate output message. Using the content of this message, the user formulates his next job step, and in this manner cyclic interactions continue until the user leaves the system. A major goal of such designs is to encourage users to interact with data and programs. If interaction is very slow or cumbersome, effectiveness will diminish. Short requests usually receive high priority through an interrupt scheme that allows the central processor to switch to a new task whenever the active one is delayed or exceeds a maximum processing threshold called a quantum interval. In this way the central processor divides its capacity among tasks awaiting execution. When a user submits a request that will require minutes, or even hours, of central processing time, interactive response should not be greatly affected. The user with a long task must realize that due to resource sharing with interactive requests, his job will take longer on a shared system than on a batch system of equal capacity. A "good model" must predict both fast response time for short jobs and response degradation for long ones.

Another observed phenomenon of time-sharing is non-linear degradation of response time as a function of system load. Systems can provide good response only within a limited range of input demand. If demand exceeds this range, response time deteriorates rapidly. Because of this degradation, many systems arbitrarily limit the number of users who are allowed to interact simultaneously with the computer. Non-linear response to increasing demand is another physical observation which should be a derived consequence of a "good model".

Computing structures allowing frequent task switching and quanta interruptions add overhead time to that already present in the basic operating system. This addition arises because of many bookkeeping functions required

to maintain status lists of tasks and shared resources. A "good model" of time-sharing systems should explicitly consider overhead degradation.

The random nature of actual quantum intervals is often ignored in analytic models. Input/output requests, paging demands in systems with virtual memory, supervisor calls, and external interrupts are causes of quantum ends, in addition to task completions and quantum overruns that combine to make actual quanta random. The following statistics from a user session at Carnegie-Mellon University on an IBM 360/67 demonstrate that the IBM time-sharing monitor, TSS, processes interrupts occurring most frequently for reasons other than task completion or excessive central processor utilization during an interaction. Figure 1.2 is a state transition diagram illustrating the results of a probe of a typical user session.

Numbers on the figure are frequencies of events. During this 16 minute probe, 927 separate interactive job steps generated 30,856 interruptions to normal processing, an average of approximately 33 interrupts per interaction. Only 367, or about one percent, of these interrupts were processing quantum overruns. A random event such as a reference to part of a program not currently in core (a page fault) triggered the vast majority of quantum aborts.

The software monitor used to gather these statistics creates an output record for every internal system event of interest. The analyst initializes a particular probe by informing TSS which events are to be traced, and the system saves the resulting output on magnetic tape for later statistical analysis. Deniston (1969) describes the design and performance characteristics of this type of measurement technique.

The preceding review summarized a number of important features of operational time-shared systems. The next section indicates the kinds of structures currently in use to model them. The highly variable nature of time-sharing

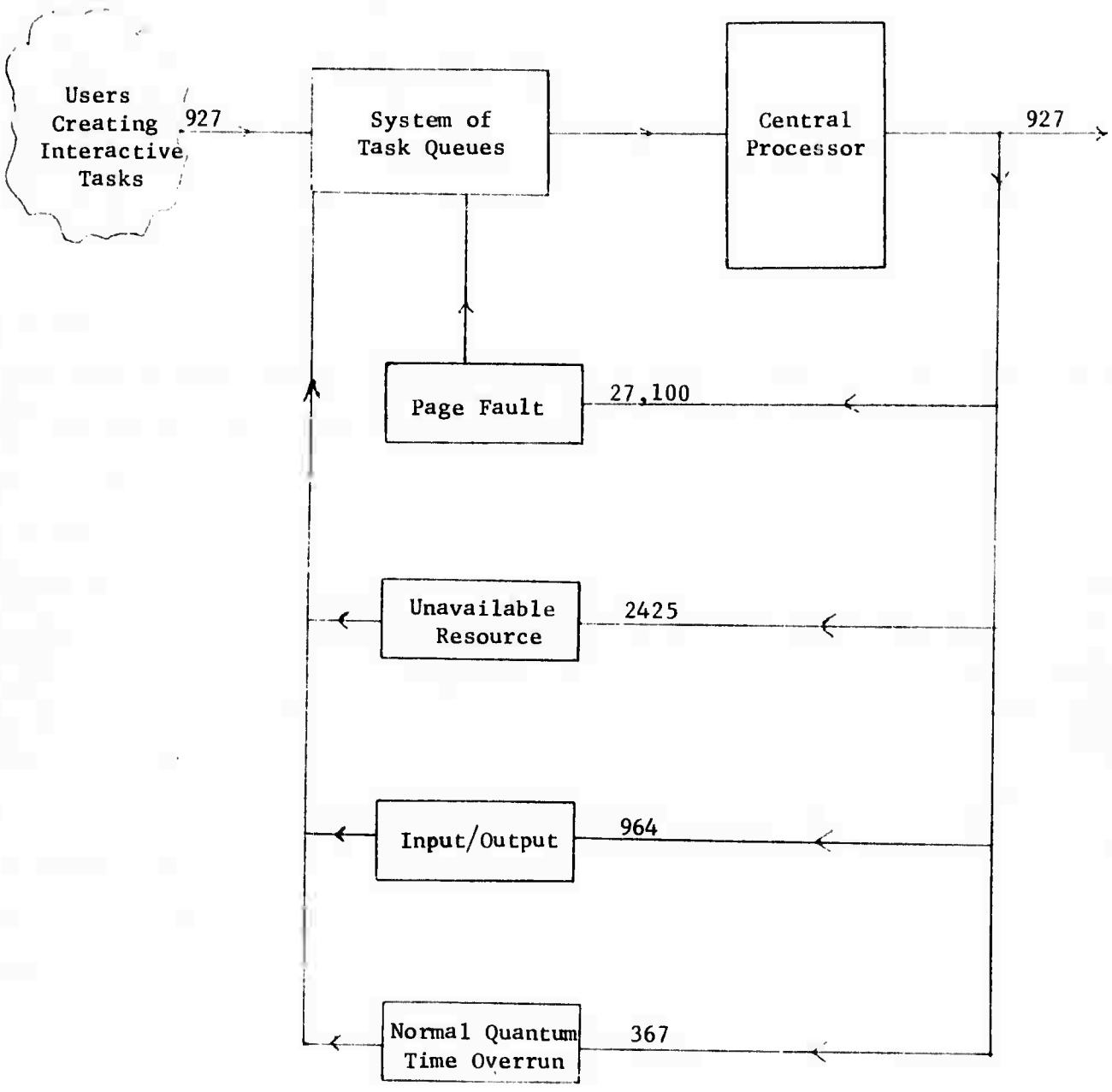


Figure 1.2

Empirical Study of Interrupt Types on TSS/360

interactions indicates that probabilistic methods should form a basis for system analysis. Queueing formulations often become very complicated even though the models are easy to describe. Thus one must carefully select areas to study within a system or queueing theory will be of little help.

1.3 A SELECTED REVIEW OF ANALYTIC TIME-SHARING MODELS

Time-sharing models have grown at a rate paralleling that of actual systems. McKinney's (1969) survey and annotated bibliography, containing 35 references, categorizes most contributions through 1969. An earlier paper by Estrin and Kleinrock (1967) presents a useful taxonomy of analytic models and a review of simulation and measurement studies of several systems. These references are excellent introductions to the general area of time-sharing models. The more limited goal of this section is to trace the development of models upon which the work of Chapter 2 depends.

Figure 1.1 may be used as a conceptual framework to classify many models. The user subsystem generates tasks for the computing subsystem. There are two common ways of modeling the input process. The most common approach is to assume that requests arrive at the computer according to a homogeneous Poisson process[4] with arrival rate λ jobs per unit time. This assumption is equivalent to stating that interarrival times between requests have an exponential distribution with mean $1/\lambda$ time units. This model of the arrival process also assumes that the input rate is independent of the behavior of the computer subsystem. The common queueing terms for this assumption are the "exponential, infinite source" input, or the "Poisson source".

The other common approach to the arrival process is to assume a finite number of independent users, each of whom submits a task and waits until it

[4] Section 2.2 contains a summary of many of the properties of a Poisson process.

has been satisfied before submitting another. For this case "think-time", commonly defined as the interval between response to one task and submittal of the next, for each of the users, has an exponential distribution with mean $1/\lambda$ time units. For this "finite source" model the combined arrival rate to the computer depends upon the number waiting for service since a user may submit only one task at a time. This structure is self balancing since the input rate decreases as the system becomes overloaded.

The exponential distribution is central to most analytic time-sharing models. If time between events is distributed exponentially, and an event has not occurred for t time units, the time remaining until the next event has the same exponential distribution as the original inter-event interval. This memoryless, or Markov, property permits many simplifications in model structure because state information concerning elapsed time since a prior event is unnecessary.

Four empirical system studies support the approximate exponential shape of interarrival time distributions, but the measurements usually have higher variance than predicted by the exponential (Totschek, 1965; Coffman and Wood, 1966; Bryan, 1967; Scherr, 1967). Although the exponential does not fit the data exactly, the additional complexity introduced by allowing general inter-arrival distributions is not justified for models having simple results as a major goal.[5] Greater input variance causes slightly increased system congestion.

There are also two common ways of modeling the basic service philosophy of a time-sharing organization. The first, and more realistic, is the "round robin model" in which one explicitly considers a quantum interval during which a single task receives all the power of the central processing unit.

---

[5] See Saaty (1961), Chapters 9 and 10, for formulations of queueing systems having generalized input processes.

If the task does not terminate naturally during this interval, it is interrupted and forced to rejoin the queue of waiting users while some other task gains access to the processor. Some models include priority levels in the queueing subsystem. The terms "processor-shared" or "pure time-sharing" denote the second basic service approach which may be viewed as a limiting case of the first method. At each instant, the fixed processor capacity, C instructions per time unit, is uniformly shared by all active tasks. Each of M active jobs receives C/M units of computing power per unit time. In the limit, as the quantum interval approaches zero, the finite quantum, round robin model becomes the processor-shared system.[6]

One of the early works to consider a feedback queueing structure similar to Figure 1.1 was the paper by Takacs (1963). His problem arose in studies of the theory of telephone traffic, and there is no mention of either time-sharing or computer system design. His formulation includes an infinite exponential source, a finite and random quantum interval, and total service times which are the sum of a geometrically distributed number of quanta each of which is drawn from the same generalized distribution. He solved this model for the expected number of tasks in the system and for the unconditioned moments of response time for a request. (An interesting aspect of the study was the use of a symbolic differentiation computer program to find the complicated expression for the second moment of response time.)

Chang (1966) realized the applicability of the Takacs work to the time-sharing domain and redefined a number of parameters to be consistent with computer terminology. He extended the original model to consider a random selection of the quantum distribution, but he solved the extension only for

---

[6] Coffman and Kleinrock (1968) summarize many of the models of these service disciplines.

the generating function of the number of queued tasks at the end of a quantum or at the instant of a job departure. Neither author calculated mean response conditioned on the amount of service requested. This latter quantity indicates how a time-sharing design will respond to tasks requiring different amounts of computing time. Section 2.3 presents a logical extension of the Takacs work in which random overhead is added to the quantum interval and mean response conditioned on service is calculated.

Kleinrock (1964) first calculated expected response conditioned on service for a simple model. He constrained events to occur only at discrete points of time corresponding to constant service quanta of length Q. At the end of each interval a new task enters the first-in-first-out (FIFO) queue with probability λQ; the job being served, if the system is not empty, either completes service (probability $1-\sigma$) or rejoins the end of the queue (probability σ); and the processor takes the job at the head of the queue for a service interval of length Q. Call the processing requirement of a job, V. A job having $V = nQ$ units is forced to join the end of the queue n times before its processing is complete. Kleinrock calculates the steady state expected number of tasks in the system, E(M) as given in equation (1.1). He also calculates the conditional response time for a job requiring nQ units of processing, $E(R|nQ)$, and shows that a good approximation to the latter result is the simple formula of equation (1.2).

$$(1.1) \quad E(M) = \frac{\lambda Q \sigma}{1-\sigma-\lambda Q}$$

$$(1.2) \quad E(R|nQ) \approx nQ\left(1 + \frac{\lambda Q \sigma}{1-\sigma-\lambda Q}\right)$$

In a later paper Kleinrock (1967) considers the limiting case (as $Q \to 0$) of the above model. For this processor-shared case the arrival process

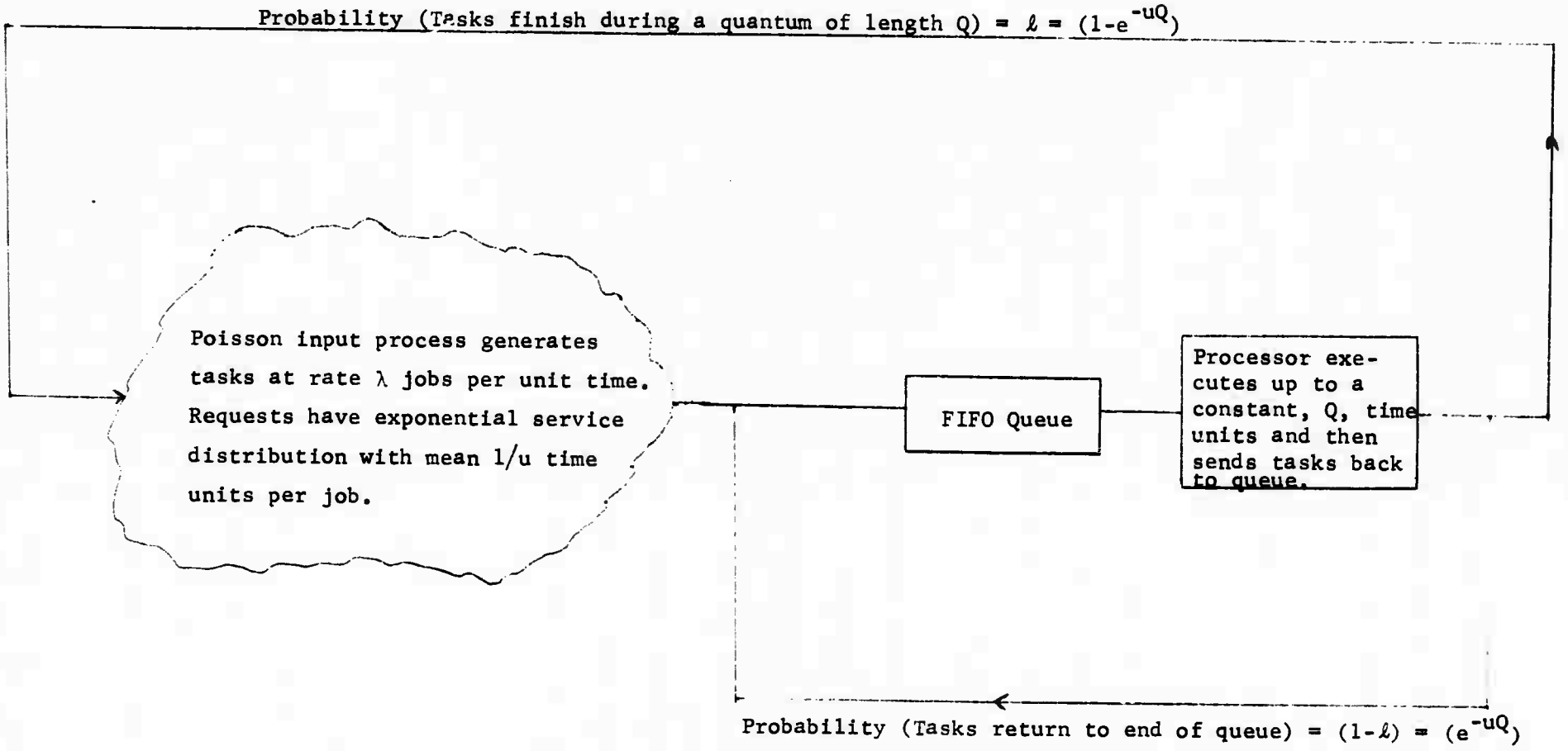
becomes the infinite exponential source with rate λ jobs per unit time, and the service requirement for each task becomes an exponential random variable with mean 1/u time units per job. Equation (1.3) is his result for expected response time conditioned on a processing need of V time units. In the concluding sections of the paper Kleinrock further extends his earlier work by considering priority classes in the queueing structure.

$$(1.3) \quad E(R|V) = V \cdot u/(u-\lambda)$$

A series of four similar articles closely related to Kleinrock's work began with a paper by Shemer in 1967. In these models computing requests come from an infinite exponential source having rate λ jobs per unit time. Processing requests are exponentially distributed random variables having an expected value of 1/u time units. Each request joins the end of a first-in-first-out queue upon arrival. Each task receives a maximum processing quantum interval of Q time units, where Q is a constant. If a request completes service before its time limit expires, it leaves the system and the processor immediately begins work on the task at the head of the queue. If a job cannot complete service during a quantum, it is interrupted and forced to join the end of the queue while the processor works on the next waiting task. Figure 1.3 illustrates this specific form of the general structure of Figure 1.1.

For this model, the expected number of tasks in the system, and the expected unconditional response time, are identical to the results for the classical Poisson source exponential service, single channel queueing system (M/M/1).[7] Shemer (1967) uses these facts in his derivation of expected

[7] In queueing literature, models are often classified using Kendall's notation: a/s/n where "a" denotes the type of arrival, "s" the type of service, and "n" the number of service channels. In the example above "M" denotes Markov, or exponential, arrival and service.



Note: Since service requests are exponentially distributed, the probability that a task will terminate service during a quantum of length Q is independent of how many quanta it has already received. The probability that an exponential random variable with mean 1/u will be less than, or equal to, Q is $(1-e^{-uQ})$.

Figure 1.3

Specific Structure of a Number of Time-Sharing Models

response time conditioned upon a particular task's service request, but he makes two errors. Using expected value arguments he derives a recursive expression for the mean time spent in queue waiting for service quantum i as a function of the expected wait for service quantum (i-1). The derivations are clear to follow and correct except for i=1 and i=2. The two properties of the model that Shemer does not treat correctly are:

1. The remaining time, Q_r, of a quantum interval in progress, if the processor is not idle, when a new task enters the system has a distribution that is different from the distribution of a full quantum interval.

2. The conditional probability that a task will return for more service, given that it has already completed part of a service quantum, is different from the probability that a task just starting a quantum interval will return for additional processing.

These errors propogate through all values of i and distort the final result. Section 2.2 contains a discussion of the properties of a quantum in progress when a new task arrives and Appendix A contains a correct derivation of this model with a slight extension. Shemer's paper concludes with extensions to the basic model involving priorities.

Coffman and Kleinrock (1968) use a slightly more complex approach to study the same model in a paper which also contains a number of interesting extensions including priority scheduling policies. Although both articles were published in the Journal of the Association for Computing Machinery, the latter paper did not indicate the errors in the earlier work. Coffman and Kleinrock made a small mistake in the derivation of the second moment

of the processing time actually received by a task during a quantum. Their result is distorted also since this intermediate formula appears in the final expression. Appendix A presents the corrected result for the second moment of a quantum interval of this type.

Adiri and Avi-Itzhak (1969) solved a model similar to Figure 1.3 with the addition of a constant delay, d, before every processing quanta. This extension adds realism to the basic model, and complicates the solution. The delay represents constant overhead degradation, or set-up time, required by the processor to switch from one task to the next. The added complexity of the solution arises from the fact that the total processor time required by a task is no longer exponential. A task needing V units of processing, where V is an exponentially distributed random variable, now requires $V + [V/Q] \cdot d$ processing units where [X] denotes the largest integer greater than or equal to X. The results for the expected number of tasks in the system and the expected unconditional response time are now identical to the exponential source, general service time, single channel queueing system (M/G/1). The authors employ sophisticated mathematical techniques involving complicated Laplace transforms and generating functions to solve for response time conditioned on service request. The solution is correct and reduces to the results of Shemer (1967) and Coffman and Kleinrock (1968) when the delay, d, is set to zero, and when the corrections noted in the preceding paragraphs are incorporated in the earlier derivations. To derive expected value of response conditioned on service, one may use expected value techniques employed by the previous authors rather than the more involved methods used by Adiri and Avi-Itzhak. Since these simpler techniques are the basis for the derivations in the following chapter, Appendix A contains a proof that the results from the two methods are identical.

Rasch (1970) studies the model of Figure 1.3 and then extends it in almost the identical manner as the work described in the preceding paragraph. He adds a constant delay d after, rather than before, each processing quantum. His approach contains a major mathematical error not present in any of the other three works. While deriving the expression for expected waiting time in the queue before receiving service quantum i, he mistakenly presumes that the mean value of all waits after the first will be the same. Although the differences are small, and one may wish to make this simplifying assumption to achieve simpler final expressions, one must realize (as the other authors did) that the wait for service quantum i depends in a non-trivial way upon the wait for service quantum(i-1).

The preceding paragraphs place into perspective previous studies that are in the direct line of development of the derivations of Section 2.3 and 2.4. The work of Section 2.5 has different historical roots since it is a finite exponential source, processor-shared, model including overhead loss as a function of system state. Early work in this area is categorized in queueing literature as "the machine interference" problem.[8] Sherr (1967) recognized the applicability of this work to the time sharing domain and presents an exponential, finite source, processor-shared model. He compares his results with a simulation and with measurements taken from the CTSS time sharing system at MIT. To approximate an overhead loss of X percent, he simply reduces the capacity of the processor by X percent.

Attempts to solve the structure of Figure 1.3 with modifications of a finite number of exponential input terminals and a constant delay d before each quanta started with the paper by Coffman and Krishnamoorthi (1964).

[8] See the book by Saaty (1961), pages 323-333, for an excellent review of results of classic work on machine interference.

This work was expanded by Krishnamoorthi and Wood (1966) and then further refined with the correction of an error by Adiri and Avi-Itzhak (1969b). Results of these studies are complicated in both the methodologies and the form of the expressions. Greenberger (1966) made a number of mathematical approximations to achieve simpler, although approximate, results. Applying a cost function to service delays he investigates the optimal size for the quantum parameter Q. In all of these studies overhead delay, d, and maximum quantum size, Q, are constants and service and input distributions are exponential.

A number of other papers present surveys of analytic models concerned with features such as externally assigned priorities and service disciplines which are dependent on system state. In addition to the two surveys referenced at the beginning of this section, and the papers already discussed, the interested reader is directed to studies such as Coffman (1966), Coffman and Muntz (1969), and Schrage (1967) and (1969).

A different modeling approach is based upon the work of Jackson (1963) and Gordon and Newell (1967). In these models tasks circulate among a number of service stations. Buzen (1971) applies these methods to multiprogramming systems, and Moore (1971) applies them to time-sharing designs. Courtois (1971) applies results of Simon and Ando (1961), concerning the dynamics of nearly decomposable systems, to queueing systems. His methods significantly simplify the numerical work required to solve hierarchical queueing networks.

Chapter 2 contains a number of models which extend the work reviewed in this section. The aim of these derivations is to include as many features of real systems as possible in model formulations which lead to straightforward results.

CHAPTER 2

NEW ANALYTIC MODELS OF TIME-SHARED COMPUTER SYSTEMS

2.1 INTRODUCTION

The analytic models presented in this chapter are the end products of compromises designed to include a number of important characteristics of current time-shared computer systems often ignored in other analytic models, but still to insure that the results are easy to understand and compute. They focus on the mean time required for a time-sharing system to respond to a user's request.

The survey of the analytical modeling literature in the previous chapter revealed that overhead and swapping times are often neglected in models of time-sharing. If considered, they appear almost exclusively as constant delays either before or after each service interval. Similarly, even though most time-sharing systems have many different quantum sizes based on a job's priority, its recent history, and the system state, models in which the quantum interval is a parameter usually consider it to be a constant and not a random variable. Results of many of these models appear as Laplace transforms which often require numerical inversion. Transforms are also used directly to obtain moments of distributions by differentiation, but often the results are very complicated. (See the following papers for illustrations of these statements: Greenberger, 1966, Krishnamoorthi and Wood, 1966; Coffman and Kleinrock, 1968; Adiri and Avi-Itzhak, 1969, Rasch, 1970.)

The new models presented in this chapter extend previous work. The first formulation includes features such as:

1. a random part of each quantum interval is required for overhead functions

2. the service/execution segment of each quantum is a random variable

3. the total service request for a task may have any distribution which can be represented as a geometrically distributed sum of independent random variables, each of which has the same arbitrary distribution

The second model is a tandem service structure which models the multiprogramming aspects of many systems by representing the behavior of a task as a random number of cycles through both a central processor and an input/output subsystem. Processing may occur simultaneously in each subsystem. The third model is a finite source system having a processor-shared service facility with overhead degradation which is a function of the state of the system.

The following symbols will be used in a standard manner throughout the report. Other notation will be introduced as needed in each section.

$P(\cdot)$ = the probability of event $(\cdot)$

$P(G|H)$ = the conditional probability of event G given event H

$F_X(t)$ = the cumulative probability distribution function of a random variable X

$= P(X \leq t)$

$= \int_{-\infty}^{t} dF_X(t)$ (the Stieltjes integral)

$f_X(t)$ = the density function of a continuous random variable X

$E(X)$ = the expected value of a random variable X

$= \int_{-\infty}^{\infty} t dF_X(t)$

$E(X|Y)$ = the conditional expectation of X given Y

$E(g(X)) = \int_{-\infty}^{\infty} g(t) dF_X(t)$ = the expected value of a function $g(\cdot)$ of a random variable X

$VAR(X)$ = the variance of a random variable X

$= E((X - E(X))^2)$

$SD(X)$ = the standard deviation of a random variable X

$= (VAR(X))^{\frac{1}{2}}$

$L_X(s)$ = the Laplace transformation of a non-negative random variable X

$= E(e^{-sX}) = \int_0^{\infty} e^{-st} dF_X(t)$

$L_X^{-1}(t)$ = the inverse Laplace transformation

Before proceeding to the derivation of the models, the next section will review some important properties about the Poisson process that are used throughout the chapter.

2.2 THE POISSON PROCESS

This section contains a brief summary of a number of well known properties of the Poisson process which are used throughout the field of queueing theory. These results are usually scattered throughout texts. The following books contain good discussions: Feller (1957); Saaty (1961); Parzen (1962); and Conway, Maxwell, and Miller (1967). The following presentation borrows extensively from the material contained in Chapter 8 of Conway, Maxwell, and Miller (1967). This text also contains an excellent bibliography for the reader interested in pursuing the subject in greater depth.

2.2.1 Process Definition

A counting process is usually defined as an integer-valued process $\{N(t),\ t \geq 0\}$ which counts the number of points occurring in an interval, these points having been distributed by some stochastic mechanism. Here the points represent the times at which events of a specified character occurred.[1] Consider events occurring in time on the interval 0 to ∞, and for $t > 0$ define $N(t)$ to be the number of events that have occurred in the interval 0 to t (the interval is open at 0 and closed at t). Let $N(0) = 0$. Then, for a Poisson process, whatever the value of $N(t)$, the probability that during $(t, t+h)$ an event occurs is $\lambda \cdot h + o(h^2)$, and the probability that more than one event occurs is $o(h^2)$. The term $o(h^2)$ denotes a quantity which is of smaller order of magnitude than h so that $o(h^2)/h$ tends to zero as h tends to zero. The Poisson process has increments between events which are independent and stationary in time.

The following equations are derived consequences of the process definition.

$$(2.1)\quad P(N(t) = n) = \frac{(\lambda \cdot t)^n e^{-\lambda t}}{n!}, \quad n=0,1,2,\ldots$$

$$(2.2)\quad E(N(t)) = \lambda \cdot t$$

$$(2.3)\quad SD(N(t)) = (\lambda \cdot t)^{\frac{1}{2}}$$

If A is the random time between two successive events, then:

$$(2.4)\quad P(A \leq t) = 1 - e^{-\lambda t}, \quad t \geq 0$$
$$= 0, \quad t < 0$$

[1] See Parzen (1962), p. 117.

(2.5) $E(A) = 1/\lambda$

(2.6) $SD(A) = 1/\lambda$

2.2.2. Memoryless Property

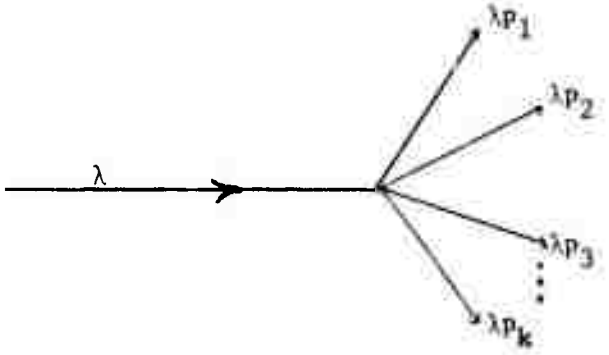
A consequence of the postulate that the probability of an event during (t, t+h) is independent of previous process history is that the time until the next event, given that no event has occurred for y time units, is independent of y.

(2.7) $P(A > y+t \mid A > y) = P(A > t) = e^{-\lambda t}, \quad t > 0, \; y > 0$

Another interesting property of the Poisson process is that if n events occur in an interval (0,t), then the n event times are independently and uniformly distributed over the interval (0,t)

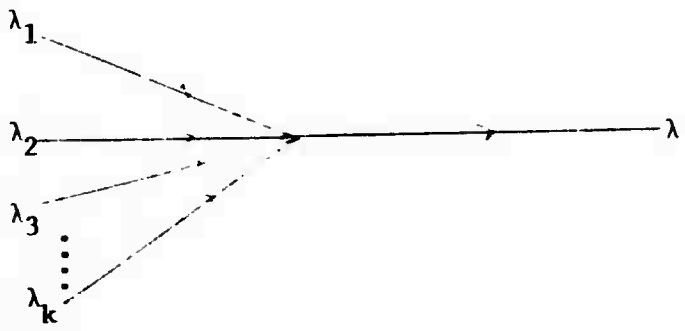
2.2.3. Branching and Aggregation of Poisson Processes

Consider a Poisson stream of events with rate λ which is randomly split into k different streams in which the probability that path i will be taken is p_i. If the output paths are chosen independently, then the i^{th} path is a Poisson stream with rate λp_i.



$$\sum_{i=1}^{k} p_i = 1$$

Conversely, if k independent Poisson streams, having rates $\lambda_1, \lambda_2, \ldots \lambda_k$, are aggregated, the resulting stream is Poisson with rate $\lambda = \sum_{i=1}^{k} \lambda_i$.



2.2.4 Remaining Service Time Distribution

Let jobs arrive at a server from a Poisson source. Given that a new job arrives while another one is being served, the remaining service time, Q_r, is defined as the time interval from the arrival of the new job until the service completion of the one already there. If service intervals, X, are random variables with distribution function $F_X(t)$, then Conway, Maxwell, and Miller (1967) derive the following two results about Q_r, the remaining service time:[2]

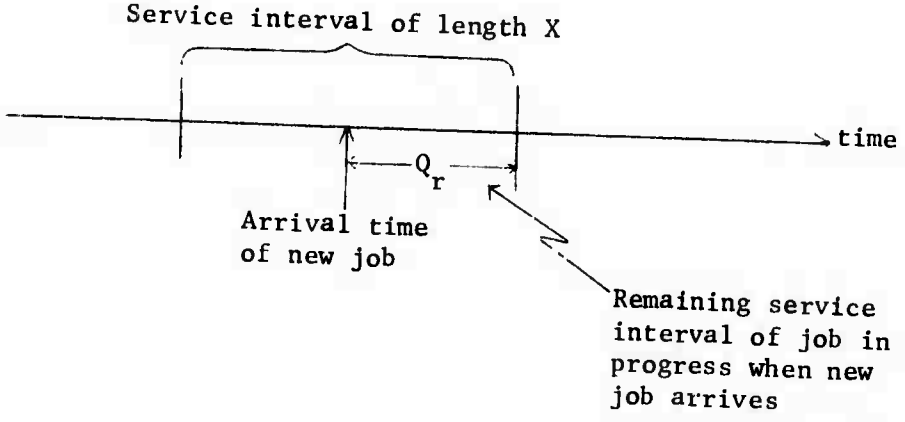
$$(2.8) \quad P(t \leq Q_r \leq t + dt) = \frac{(1 - F_X(t))dt}{E(X)}, \quad 0 \leq t \leq \infty$$

$$(2.9) \quad E(Q_r^k) = \frac{E(X^{k+1})}{(k+1)\ E(X)}, \quad k=1,2,\ldots$$

Equation (2.8) gives the probability that Q_r will be in a small interval, and (2.9) gives the k^{th} moment of Q_r. The following diagram illustrates

[2] Conway, Maxwell, and Miller (1967), p. 146-147. Appendix A contains derivations of equations (2.8) and (2.9). The method used to derive (2.8) differs from the approach of Conway, Maxwell, and Miller.

the relationship between X, and Q_r. The results are conditioned on there being a job in service when a new one arrives.



2.2.5 Output of an M/M/1 Queue

If the arrival to a single server queueing system is Poisson with rate λ, and the service is exponential with rate $\mu > \lambda$, and if the queue scheduling procedure is independent of the set of processing times of the jobs, then in the steady state the departure intervals are independently distributed exponential variables with parameter λ. In other words, the output process is Poisson with the same rate, λ, as the input process. This result may be extended to the generalized birth-death process, and thus it also applies to M/M/n systems.

2.2.6 Results for M/G/1 Queue

If the input process to a single server queueing system is Poisson with rate λ, if the processing time, X, has a general distribution with mean $1/\mu$, and if the traffic intensity $\rho = \lambda/\mu$ is less than one, then for any service discipline that is independent of the processing times of the jobs

(such as first-in-first-out), the following results are steady state values for important system parameters.

(2.10) E(flow time through queue and server) = E(R)

$$= \frac{2 \cdot E(X) \cdot (1-\rho) + \lambda \cdot E(X^2)}{2 \cdot (1-\rho)} = E(X) + \frac{\lambda \cdot E(X^2)}{2 \cdot (1-\rho)}$$

(2.11) P(server is idle) = $1-\rho$

Equation (2.10) is the classic Pollaczek-Khintchine formula. The expected number of jobs in the queue and in the server is related to this result by (2.12), Little's theorem (1961). These results show that different processing time independent scheduling rules have no effect on the mean number in the system, the mean response time, and the probability that the server will be idle. Scheduling rules which are independent of processing time do have effects on the response times of individual jobs, but not on the expected value of response for all jobs.

(2.12) E(number of tasks in M/G/1 system) = $\lambda \cdot E(R)$

2.3 A TIME-SHARING MODEL WITH RANDOM QUANTA AND RANDOM OVERHEAD - TSMOD1

The model in this section has the following basic structure: a Poisson source of tasks; a random delay drawn from a general distribution representing overhead loss due to quantizing; a random processing quantum drawn from an arbitrary distribution; and feedback to a round robin, first-in-first-out queue. The independent variables are: speed of the processor; interrupt probability; and the probability distributions of quantum requests, overhead delays caused by each interrupt, and interarrival

times between requests. The dependent variable is the expected value of the time required by the system to service a request. Although analysis becomes complicated, the directly applicable result is simple. This model is useful since it retains simplicity while including a number of essential parameters for time-shared systems. Figure 2.1 illustrates the structure of the model.

2.3.1 Definitions and Model Formulation

Define the following symbols for use in the model.

C = a constant equal to the computer processing rate expressed in instructions per unit time

ℓ = the probability that a task has been completed after an interrupt

W = the random number of instructions executed during a processing quantum before an interrupt occurs

wi = the i^{th} moment of W

V = a random variable denoting the number of instructions required by a task for one complete interaction

vi = the i^{th} moment of V

D = a random variable representing the overhead delay of an interrupt

di = the i^{th} moment of D

M = the random number of tasks in the queue and in the server

R = response time, a random variable denoting the elapsed time from task submittal to task completion (queue waiting time plus service and overhead time)

N = the number of interrupts experienced by a task during the execution of its V instructions

A = a random variable expressing the interarrival time between tasks requesting service from the system

Q = the random length of a quantum interval which is the sum of a service segment, W/C, and an overhead delay D.

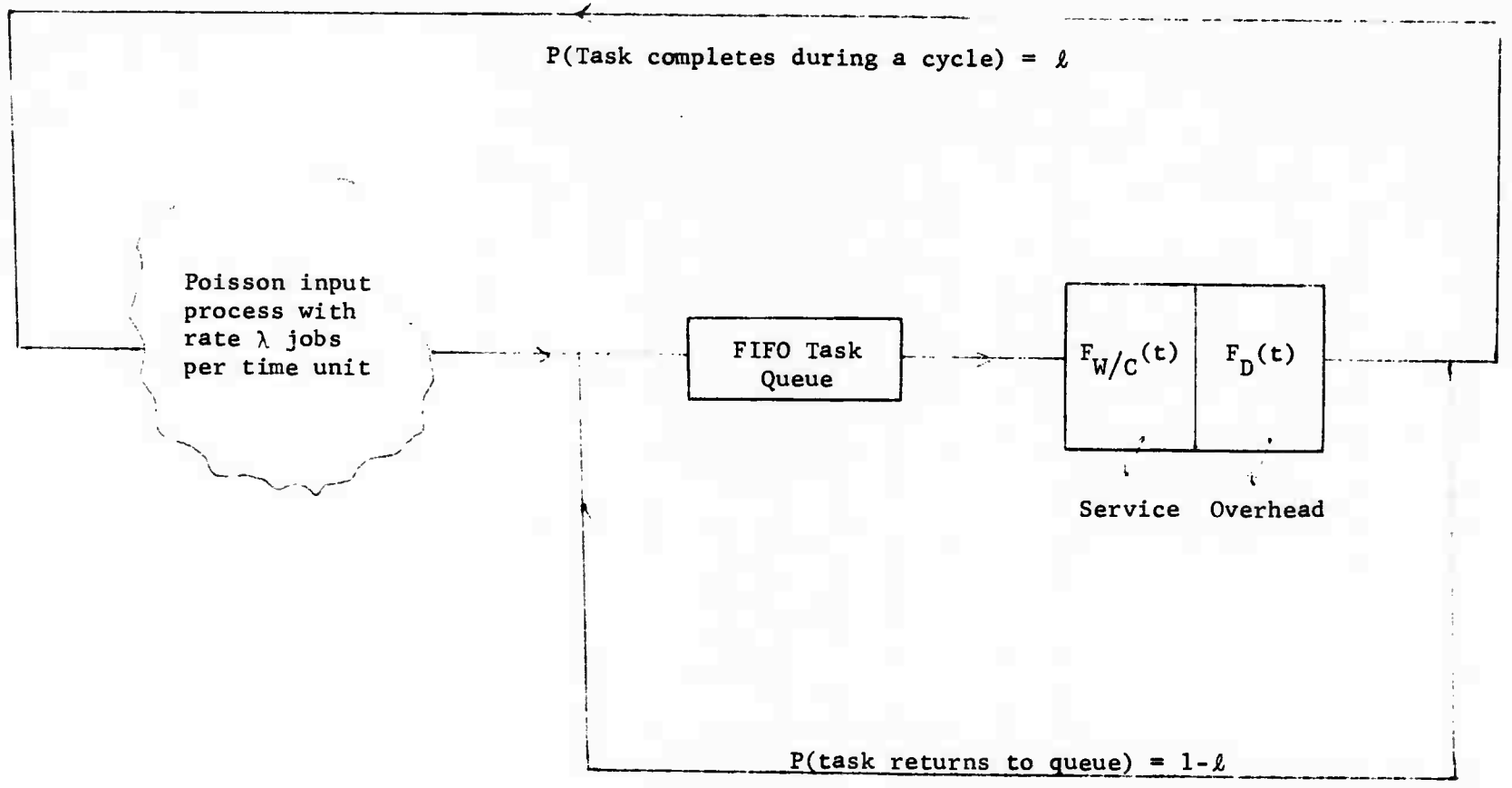


Figure 2.1

Structure of Model with Overhead - TSMOD1

The original model presented by Takacs (1963) has the following formulation and solution within this more general framework. Requests for service arrive at the system from a Poisson source. If e_i is the time the i^{th} job enters the system, interarrival times are $A_i = e_i - e_{i-1}$. All A_i are independent, identically distributed random variables having an exponential distribution with expected value $E(A) = 1/\lambda$. (See equations 2.4. and 2.5.)

Models based on the simplistic assumption of constant quanta are neglecting a primary feature of many real systems. A way of approximating the fact that tasks often return to the queue after using only a small amount of the maximum allowable quantum (for example, to wait for an input/output request) is to make the quantum W a random variable. The server works on tasks in a cyclic, round robin manner. After each task receives a random service quantum it either leaves the system (probability ℓ), or rejoins the end of the queue (probability $1-\ell$) while the processor works on the next task. The event, "job rejoins the queue", is independent of both the length of quantum service, and the number of quanta the job has received. The distribution of N, the number of job interruptions, is geometric with expected value $1/\ell$.

$$(2.13)\quad P(N=n) = \begin{cases} \ell(1-\ell)^{(n-1)}, & n=1,2,3,\ldots \\ 0, & \text{elsewhere} \end{cases}$$

W may have a general distribution function with the obvious restriction that it be non-negative. The first three moments of W are w1, w2, w3. The total service request for a complete interaction, V, consists of the sum of a random number of random quanta W_i.

$$(2.14)\quad V = \sum_{i=1}^{N} W_i$$

N has the probability mass function specified in equation (2.13). Equation (2.15) is the Laplace transform of V in terms of the Laplace transform of W.

$$(2.15)\quad L_V(s) = \sum_{i=1}^{\infty} \{L_{W_1+\ldots+W_i}(s) \cdot P(N=i)\}$$

$$= \sum_{i=1}^{\infty} (L_W(s))^i \cdot \ell \cdot (1-\ell)^{i-1}$$

$$= L_W(s) \cdot \ell / \{1-L_W(s) \cdot (1-\ell)\}$$

Moments of a distribution may be calculated from its Laplace transform by differentiation.

$$(2.16)\quad E(V) = v1 = -\left.\frac{dL_V(s)}{ds}\right|_{s=0} = w1/\ell$$

$$(2.17)\quad E(V^2) = v2 = \left.\frac{d^2L_V(s)}{ds^2}\right|_{s=0} = \frac{\ell \cdot w2 + 2 \cdot (1-\ell) \cdot (w1)^2}{\ell^2}$$

From this viewpoint total service time per interaction, V, is determined by ℓ and individual quantum times W_i. V may have any distribution that can be represented as the geometric sum of variables having an arbitrary distribution $F_W(t)$.

For the remainder of this section, consider C, the processing rate of the computer, to be one instruction per time unit so that W and V

are measures of the quantum and interaction times, as well as the numbers of instructions executed. (Quantum time = (number of instructions per quantum)/(processing rate of computer)). Using arguments that relate this model to the M/G/1 queueing system, Takacs (1963) establishes the following formulas for the steady state mean value, and second moment, of response time R.

$$(2.18)\quad E(R) = \frac{\lambda w2 + 2w1(1-\lambda w1)}{2(\ell-\lambda w1)}$$

$$(2.19)\quad E(R^2) = \frac{\ell^2 - 2\ell}{6(\ell-\lambda w1)^2[\ell^2 - \ell(2+\lambda w1) + \lambda w1]} \cdot \{2\ell[6\lambda w1^3 - 6w1^2 - 6\lambda w1w2 + 3w2 + \lambda w3] - [12\lambda w1^3 - 12w1^2 - 6\lambda w1w2 + 2\lambda^2 w1w3 - 3\lambda^2 w2^2]\}$$

The necessary and sufficient conditions for these equations to be valid steady state solutions are that $\lambda w1/\ell < 1$ and that w2 and w3 be finite. The term $\lambda w1/\ell$ is similar to the standard definition of traffic intensity, ρ, which is the mean arrival rate divided by the mean service rate. In this model the effective arrival rate is λ/ℓ which includes tasks which are fed back from the server for additional processing. One may apply Little's Theorem (equation 2.12) to equation (2.18) to obtain the expected number of tasks in the system, E(M).

2.3.2 A First Extension to the Basic Model

The model described above may be extended in a number of directions to include more features of time-sharing. Processor speed, and interrupt overhead become explicit independent variables in the following analysis.

Previously W and V were equivalent to time units since the processing rate was one instruction per time unit. C, the constant computer speed, is expressed in instructions per unit time. W is the number of instructions processed before an interrupt occurs, and V is the total number of instructions required for a task to complete one interaction. The time spent in one service quanta will be $W' = W/C$, and total service time for one task will be $V' = V/C$.

Interrupts cause overhead. For example, systems with virtual memory structures, such as the IBM 360/67 and the GE 645, require approximately five milliseconds to process a page fault.[3] This type of overhead may be included in the model by adding an interrupt processing delay, D, to each quantum W/C. D is a random variable, independent of W, having first three moments d1, d2, and d3. The central processor continues to work in a cyclic manner, but after a quantum interval on one task, it cannot start another until the interrupt processing time D has elapsed. The addition of D defines a new total quantum time, $Q = W/C + D$ with first three moments q1, q2, and q3. Using the fact that the Laplace transform of the sum of two independent random variables is the product of the individual transforms, one may easily differentiate $L_Q(s)$ to obtain its moments.

$$(2.20)\quad L_Q(s) = L_{W/C}(s) \cdot L_D(s)$$

$$(2.21)\quad q1 = w1' + d1 = w1/C + d1$$

$$(2.22)\quad q2 = w2' + 2w1'd1 + d2 = w2/C^2 + 2w1d1/C + d2$$

$$(2.23)\quad q3 = w3/C^3 + 3d1w2/C^2 + 3d2w1/C + d3$$

[3] See, for example, the experiments performed on the MULTICS system at MIT by Corbato (1968).

Equation (2.15) still defines the relationship between W and V. The addition of D to W does not change the users' demands, but the processor takes more elapsed time to satisfy a request in the presence of overhead than without it.

Replacing w1 and w2 with q1 and q2 (equations (2.21) and (2.22)) in equation (2.18), leads to the following expression for mean response in this system.

$$(2.24)\quad E(R) = \frac{[\lambda\{w2/C^2 + 2w1d1/C + d2\} + 2(w1/C + d1) \cdot (1-\lambda(w1/C + d1))]}{2\cdot[\ell-\lambda(w1/C + d1)]}$$

The necessary and sufficient condition for existence of a steady state solution is that $\lambda(w1/C + d1)/\ell < 1$. The quantity on the left of this expression is the effective traffic intensity for the extended model. One can make the same substitution in (2.19) to investigate the behavior of the second moment of response time.

2.3.3 Two Examples

Let V, the number of instructions required to complete a task's request, have a general non-negative distribution with first and second moments v1 and v2. Consider the simplified case of no overhead and no service interruptions (d1 = d2 = 0; ℓ = 1). Since w = v, this case corresponds to batch processing where each interaction is processed to completion. Equation (2.24) reduces to (2.10), the Pollaczek-Khintchine result for average response in an M/G/1 queueing system. (The first two moments of service time are v1/C and $v2/C^2$.)

$$(2.25)\quad E(R) = \frac{\lambda v2/C^2 + 2(v1/C)(1 - \lambda v1/C)}{2(1 - \lambda v1/C)}$$

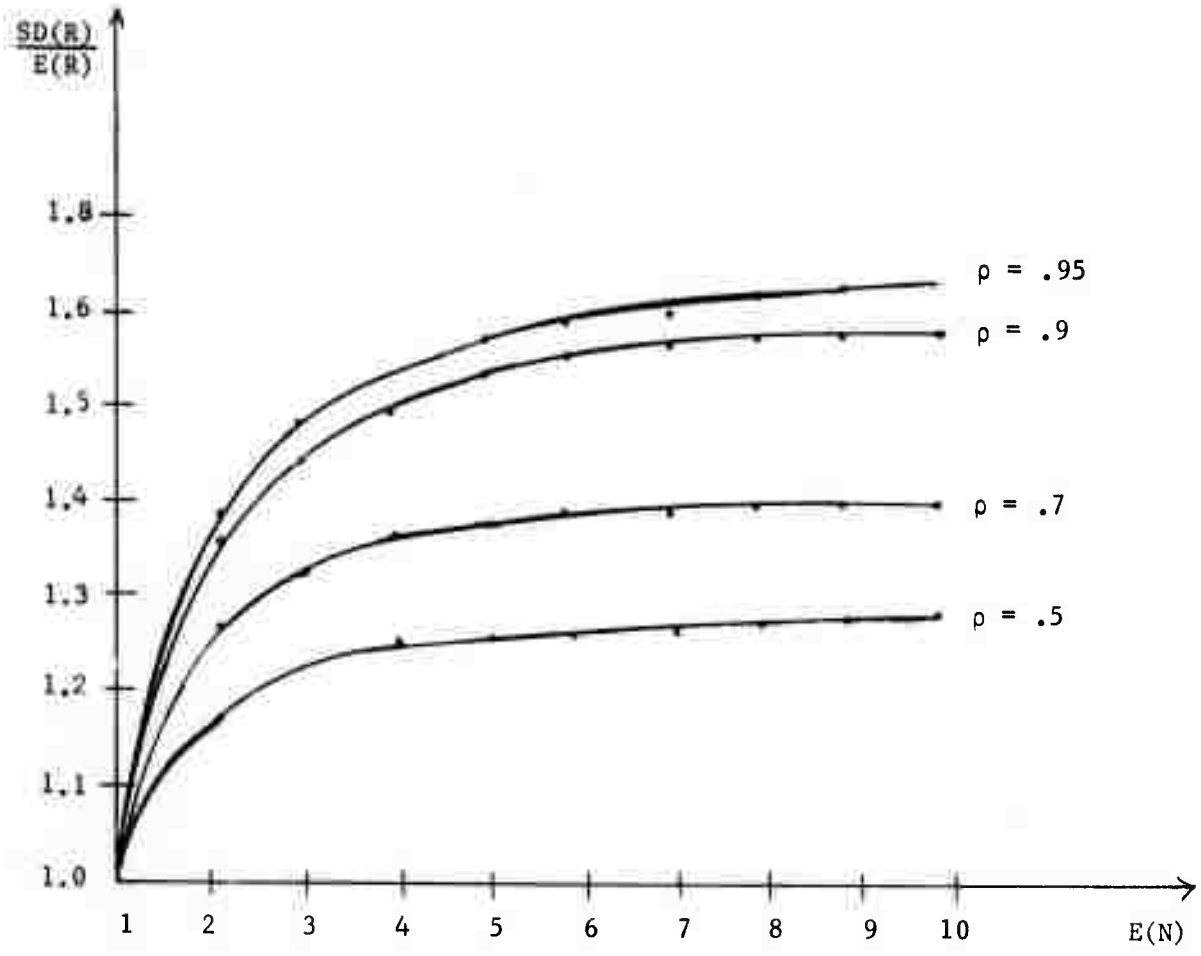
Now consider service to be quantized, but still with no overhead, D, associated with quantum interrupts. W and ℓ will be adjusted to hold the moments of V constant. The probability, ℓ, of completing an interaction after a quantum of length W is less than one. Substituting the appropriate variables from (2.16) and (2.17) in (2.24) one arrives at the following expression for the quantizing model without overhead. The final form is identical to (2.25).

$$(2.26) \quad E(R) = \frac{\lambda\{\ell v2 - 2(1-\ell)\ \ell v1^2)/C^2\} + 2\ell(v1/C)(1 - \lambda\ell v1/C)}{2(\ell - \lambda\ell v1/C)}$$

$$= \frac{\lambda v2/C^2 + 2(v1/C)(1 - \lambda v1/C)}{2(1 - \lambda v1/C)}$$

Equation (2.26) is unintuitive since, for this model, quantizing without overhead has no effect on E(R), the expected value of response time. Any overhead will increase E(R). If quantizing does not improve mean response, and actually degrades it due to overhead, one may reasonably ask what benefit accrues from this scheduling policy. Briefly, the benefit is that short requests receive better than average response at the expense of long requests. The policy of favoring short interactive requests penalizes longer tasks and degrades overall response when there is overhead associated with quantum interrupts.

Figure 2.2 illustrates that even though mean response remains constant as more overhead-free quantizing occurs, the standard deviation of response increases. For this example each quantum is exponentially distributed. The total service request, V, is also exponential since a geometric sum of exponentials is an exponential. The mean of V is held



E(N) = 1/ℓ	ρ = .5	ρ = .7	ρ = .9	ρ = .95
1	1.000	1.000	1.000	1.000
2	1.183	1.268	1.363	1.388
3	1.225	1.333	1.458	1.492
4	1.243	1.363	1.502	1.541
5	1.254	1.380	1.528	1.569
6	1.260	1.390	1.544	1.587
7	1.265	1.398	1.556	1.601
8	1.268	1.404	1.565	1.610
9	1.271	1.408	1.571	1.618
10	1.273	1.411	1.577	1.624

Each table entry is the ratio of the standard deviation of response to the expected value of response.

Figure 2.2

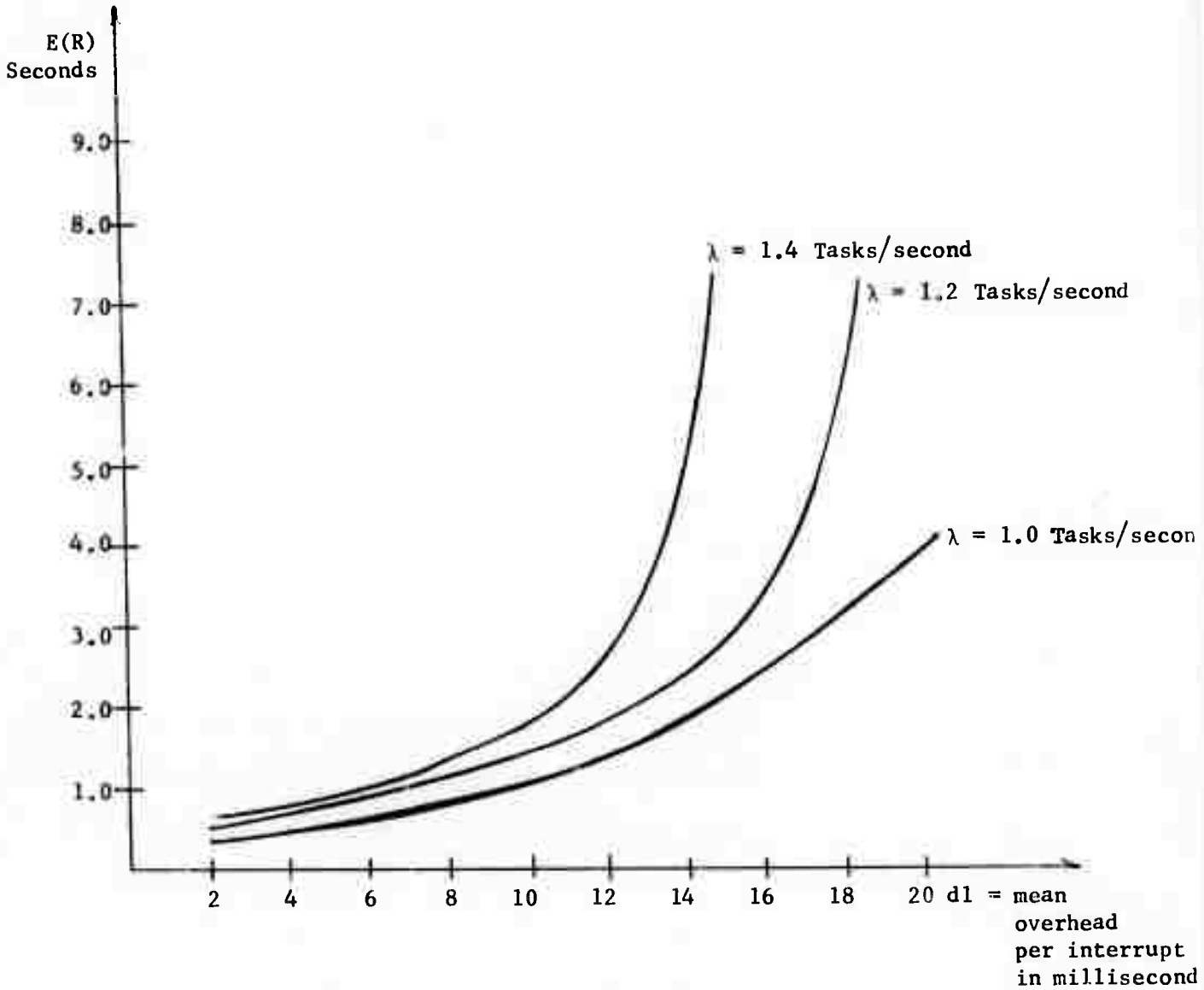
Effect of Quantum Size on Standard Deviation of Response - SD(R)

constant and the mean quantum size and the mean number of interruptions ($1/\ell$) are varied. The input rate increases to examine the effect of increasing the user demand. For each value of λ, mean response remains constant. Increased service variability is another undesireable effect of round-robin scheduling which must be balanced by increased responsiveness to short requests.

The effects of overhead on response are obvious in the second example. Let processor speed, C, be 500,000 instructions per second, and the mean request for a complete task, v1 be 100,000 instructions. Let the standard deviation of the instruction requests be 150,000 instructions. The average processing time per task, v1/C, is 200 milliseconds. The probability that a task will require additional processing after an interrupt is .97 ($\ell = 1/30$). Thus a task produces an average of 30 interrupts per interaction, and the mean non-overhead quantum time between interrupts is 6.67 milliseconds. Figure 2.3 displays expected response, as a function of mean overhead delay, d1, for a number of values of λ, the Poisson arrival rate of requests. For each curve, the standard deviation of D is twice its mean of d1. These results show commonly observed response degradation caused by overhead delay and by congestion resulting from increasing arrival rates.

2.3.4 Mean Response Conditioned on Service Request

This section contains exact and approximate expressions for mean response time conditioned on measures of the service requirement. $E(R|N=n)$ is the expected response for a particular task, requiring n quanta, which will be marked and followed until it leaves the system. Let M_i be the number of tasks ahead of this tagged job (both in queue and



C = processor speed = 500,000 instructions per second

$1/\ell$ = expected interrupts per task = 30

v1 = mean number of instructions per task = 100,000

Figure 2.3

Effects of Overhead and Input Rates on Expected Response - TSMOD1

in the processor) as it enters the queue to wait for its i^{th} processing quantum. Define the i^{th} wait, T_i, to be the time the tagged job waits in queue as the M_i tasks preceding it receive their quanta.

$$(2.27) \quad T_i = \sum_{k=1}^{M_i} Q_k$$

In this equation task quantum time, Q_k, includes both the random delay D and processing time W/C.

The first cycle (i=1) is a special case since the remaining quantum interval of the task being served when the tagged job arrives has a distribution different from other quanta. Consider cycles after the first. Processing quanta of all tasks, including the marked job, have the same distribution. The expected value of the sum of m identically distributed random variables is m times their expected value. Thus the conditional expectation of T_i given $M_i = m$ is:

$$(2.28) \quad E(T_i | M_i=m) = m \cdot E(Q) = m \cdot q1, \quad i=2,3,...$$

Removing the condition by taking the expectation with respect to M_i leads to the unconditional expected value of T_i.

$$(2.29) \quad E(T_i) = E(M_i) \cdot q1; \quad i=2,3,...$$

The number of tasks in the system at the start of the i^{th} cycle is dependent on system state changes during cycle (i-1). The probability that a job will leave the system after a quantum interval is ℓ, and the probability that it will return to the queue is $(1-\ell)$. Thus the expected

number of jobs in front of the tagged job at the start of the i^{th} cycle, that was also in front of this job at the start of the (i-1) cycle, is $(1-\ell) \cdot E(M_{i-1})$. In addition, new tasks from the input process which arrive during the tagged job's (i-1) queueing wait plus service quantum will also be ahead of the tagged job as it begins waiting for its i^{th} quantum. Since the mean number of arrivals from an exponential source with rate λ during time period T is λT, the expected number of new arrivals during $(T_{i-1} + ql)$ is $\lambda \cdot (T_{i-1} + ql)$. Taking the expectation with respect to T_i leads to the following recursive expression for expected value of M_i as a function of the expected values of M_{i-1} and T_{i-1}.

$$(2.30) \quad E(M_i) = (1-\ell) \cdot E(M_{i-1}) + \lambda \cdot (E(T_{i-1}) + ql), \quad i=2,3,...$$

Specification of $E(M_1)$ and $E(T_1)$ allows one to use equations (2.29) and (2.30) to calculate all future waits. Since the arrival process is Poisson and independent of the service process, a new task arrives at a random time. $E(M_1)$ is thus the steady state expected number of customers in the system, E(M), given by applying equation (2.12) to (2.24). Calculation of $E(T_1)$ is complicated by the fact that when the tagged job arrives at a busy system, the task currently being processed has been in service for a random interval and its remaining quantum service, Q_r, is distributed differently from other quanta. As indicated in Section 1.3 Shemer (1967) did not recognize this fact and his exponential service model without overhead contains errors due to this oversight. Figure 2.4 illustrates the situation.

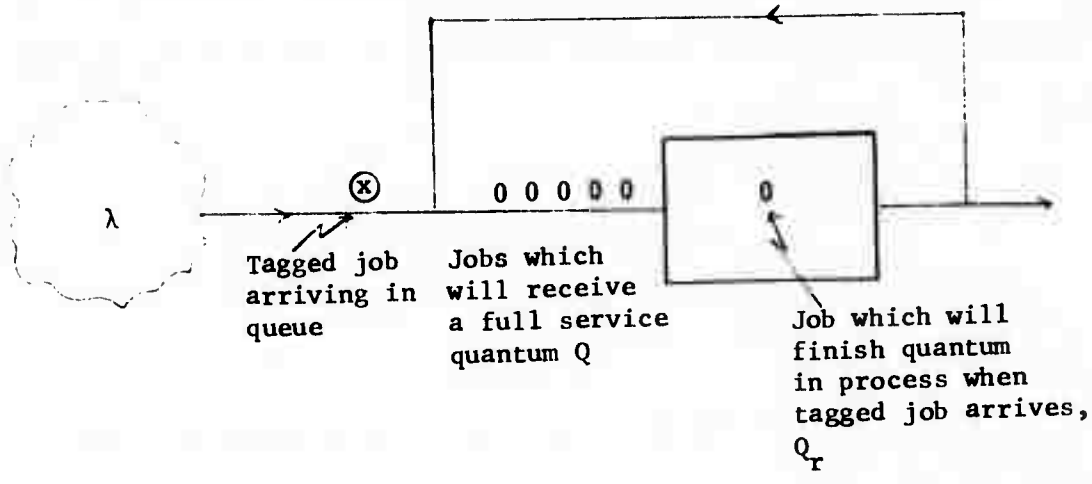


Figure 2.4

Calculation of First Wait Time, T_1

Since q1 is the expected value of a full service quantum, let $q1_r$ be the expected value of remaining quantum service, Q_r, of the job being processed when the tagged job initially arrives at the queue. Let p_i be the probability that there are i jobs in the system when the tagged task arrives. Then the expected wait in queue of this task before it begins service is the sum of mean values of the service quanta of all queued jobs and the expected remaining quantum service of the task in the processor.

$$(2.31)\quad E(T_1) = q1_r \cdot p_1 + (q1_r + q1)p_2 + (q1_r + 2 \cdot q1) \cdot p_3 + \cdots$$

$$= q1_r \cdot (1-p_0) + q1 \sum_{i=1}^{\infty} (i-1)p_i$$

$$= q1_r \cdot (1-p_0) + q1 \cdot (E(M) - (1-p_0))$$

$$= \rho \cdot q1_r + q1 \cdot (E(M) - \rho)$$

where $\rho = 1-p_0 = \lambda \cdot q1/\ell$, (equations (2.10) and (2.16))

and $E(Q_r) = q1_r = q2/(2 \cdot q1)$, (equation 2.9).

All values necessary for the calculation of expected total wait, T, conditioned on the number of required quanta, n, are now in easily computable form.

$$(2.32)\quad E(T|N=n) = \sum_{i=1}^{n} E(T_i)$$

$$= E(T_1) + \sum_{i=2}^{n} E(T_i)$$

One may express this result in closed form by using equations (2.29) and (2.30) and straightforward applications of the following identity concerning finite series.

$$(2.33)\quad 1 + x + x^2 + \ldots + x^n = (1-x^{n+1})/(1-x), \quad x\neq 1$$

$$= n+1 \quad , \quad x=1$$

$$(2.34)\quad E(M_{k+1}) = a^{(k-1)} \cdot E(M_2) + b\left\{\frac{1-a^{k-1}}{1-a}\right\}, \quad k=2,3,\ldots$$

where $a = (1-\ell) + \lambda \cdot q1$

$b = \lambda \cdot q1$

Substitution of this expression in (2.32) leads to a closed form for $E(T|N=n)$.

$$(2.35)\quad E(T|N=n) = E(T_1) + q1 \cdot E(M_2) \cdot \left(\frac{1-a^{n-1}}{1-a}\right) + \frac{b \cdot q1}{(1-a)^2} \cdot \{(n-2)(1-a) - a(1-a^{n-2})\}$$

$$n=1,2,3,\ldots$$

where $E(T_1)$ is given by (2.31)

$E(M_2) = \lambda \cdot (E(T_1) + q1) + (1-\ell) \cdot E(M)$

$E(M)$ is given by (2.24) and (2.12)

Expected total response, conditioned on N, is the sum of mean expected total wait and mean total service given that N = n.

$$(2.36)\quad E(R|N=n) = E(T|N=n) + n \cdot q1$$

Appendix A includes a derivation of the results of a different model studied by Adiri and Avi-Itzhak (1969) using the techniques of this section rather than their complicated transform methods. Their model has a Poisson source of requests, and constant swapping overhead with exponential service requests. The results of the two different types of analysis are identical.

2.3.5 A Simplifying Approximation

Equation (2.35) is not an intuitive expression. An interesting approximation is to let mean waiting time in queue for each cycle after the first, $E(T_i)$, be equal to the steady state expected number of tasks in the system, E(M), multiplied by the mean quantum interval, q1. Table 2.1 demonstrates that the magnitude of the error introduced by making this approximation is small. The exact result, equation (2.35), enables one to measure effects of such simplifying approximations. Shemer (1967) and Rasch (1970) both made approximations without realizing it and without measuring the effects. These results show that their derivations, although not exact, are close to the correct solutions.

Equation (2.31) is the exact expression for $E(T_1)$, the mean wait in queue before a task begins to receive its first service. One could use the approximation for this quantity also, but the additional complexity added by including the exact expression is small. Using the approximation $E(T_i) \approx E(M) \cdot q1$ in equation (2.32) leads to the following result for mean total wait in queue, given that N = n.

Parameters for TSMOD1

Example 1

These parameters are used in the simulation of Section 3.3.

$C = \lambda = 1$

$\ell = 1/8$

$d1 = w1 = .05$

$d2 = w2 = 2.725 \cdot 10^{-3}$

$E(M) = 3.81$

n	Equation 2.35	$E(T_1) + (n-1) \cdot q1 \cdot E(M)$
1	.343	.343
2	.720	.724
3	1.098	1.105
4	1.477	1.485
5	1.856	1.866
6	2.236	2.247
7	2.616	2.628
8	2.997	3.009
9	3.378	3.390
10	3.760	3.771
20	7.600	7.580
30	11.476	11.389
40	15.379	15.198
50	19.305	19.007

Example 2

These parameters are used in the example of Section 5.3.1.

$C = 150{,}000$ $\quad w1 = 1250$

$\lambda = 2$ $\quad w2 = 2 \cdot w1 \cdot w1$

$\ell = 1/40$ $\quad d1 = .005$

$E(M) = 3.92$ $\quad d2 = 2.6 \cdot 10^{-5}$

Equation 2.35	$E(T_1) + (n-1) \cdot q1 \cdot E(M)$
.0367	.0367
.0764	.0764
.1160	.1162
.1157	.1159
.1953	.1956
.2350	.2353
.2746	.2750
.3143	.3147
.3540	.3544
.3936	.3941
.7903	.7911
1.1872	1.1882
1.5842	1.5852
1.9813	1.9822

Table 2.1

Comparison of Exact and Approximate Expressions for $E(T|N=n)$

$$(2.37)\quad E(T|N=n) = E(T_1) + (n-1)\cdot q1\cdot E(M)$$

$$= \rho\cdot(q2/2\cdot q1 - q1) + n\cdot q1\cdot E(M), \quad n=1,2,...$$

A more interesting form of mean conditional response time is to remove the condition on N, the number of quanta received by a task, and replace it with a condition on V, the actual processing request. To remove this condition, one must determine the distribution of the number of quanta required to fulfill a processing request v.

$$(2.38)\quad E(T|V=v) = \sum_{n=1}^{\infty} E(T|V=v,N=n)\cdot P(N=n|V=v)$$

$$= \rho\cdot(q2/2\cdot q1 - q1) + q1\cdot E(M)\cdot \sum_{n=1}^{\infty} n\cdot P(N=n|V=v)$$

To evaluate the infinite summation in the above equation, one must first determine $P(N|V=v)$, the conditional probability that a task will experience N quanta given that it requires v instructions from the central processor. This summation is the conditional mean of N given that $V = v$. Define $(X) \sim f(x)$ to mean that the random variable X has the distribution given by $f(x)$. Let $f_X^{*n}(x)$ be the n-fold convolution of the random variable X $\left(\text{i.e., } f_X^{*n}(x) = f_{X_1+\ldots+X_n}(x)\right)$. The total service request for a task, V, is the sum of N independent quanta, W, where N has the probability mass function defined in equation (2.13).

$$(2.39)\quad (V|N=n) \sim f_W^{*n}(v)$$

$$(2.40)\quad (V,N) \sim f_W^{*n}(v)\cdot P(N=n)$$

$$(2.41)\quad (V) \sim f_V(v) = \sum_{n=1}^{\infty} P(N=n)\cdot f_W^{*n}(v)$$

$$(2.42)\quad P(N=n \mid V=v) = \frac{P(N=n)\ f_W^{*n}(v)}{f_V(v)}$$

$$= \frac{\ell\cdot(1-\ell)^{n-1}\cdot f_W^{*n}(v)}{f_V(v)}, \quad n=1,2,\ldots$$

One may now substitute equation (2.42) into (2.38) to calculate the final form of the mean total wait in queue conditioned on a service request of v instructions. The specific form of the result will depend on the density function of W.

2.3.6 An Exponential Service Quantum Example

A specific example will illustrate this model. The use of an exponentially distributed quantum, W, keeps the mathematics simple because:

(1) the sum of n identically distributed exponential variables, when n is a constant, is a random variable having a gamma distribution

(2) the sum of N identically distributed exponential variables, when N is a random variable having a geometric distribution, is an exponential variable.

These two well known facts may be verified by calculating the appropriate Laplace transforms and comparing them to the transforms of the gamma and exponential distributions.

Let the density function of W be exponential with mean w1. Therefore the density function of f_W^{*n} is gamma and the density function of V is exponential with mean v1 = w1/ℓ.

$$(2.43)\quad f_W(v) = \frac{e^{-v/w1}}{w1}, \qquad v \geq 0$$

$$(2.44)\quad f_W^{*n}(v) = \frac{(v/w1)^{n-1} \cdot e^{-v/w1}}{w1 \cdot (n-1)!}, \qquad v \geq 0,\ n=1,2,\ldots$$

$$(2.45)\quad f_V(v) = \frac{\ell \cdot e^{-\ell \cdot v/w1}}{w1}, \qquad v \geq 0$$

Substituting the above equations into (2.42) leads to the desired result for $P(N|V=v)$.

$$(2.46)\quad P(N|V=v) = \frac{((1-\ell)\cdot v/w1)^{n-1} \cdot e^{-(1-\ell)\cdot v/w1}}{(n-1)!}, \qquad v \geq 0,\ n=1,2,\ldots$$

$$(2.47)\quad E(N|V=v) = \sum_{n=1}^{\infty} n \cdot P(N|V=v) = \frac{w1 + (1-\ell)\cdot v}{w1}, \qquad v \geq 0$$

The conditional mass function for N, the number of quanta needed to get v instructions, is almost the standard Poisson distribution with parameter $v\cdot(1-\ell)/w1$. Note that the mean number of quanta needed to receive v instructions is not the more intuitive quantity v/w1 where w1 is the mean number of instructions received per service quantum.

Substituting (2.47) in (2.38) gives a closed form for mean total wait in queue given that a task requires v instructions.

$$(2.48)\quad E(T|V=v) = \rho\cdot(q2/2q1 - q1) + q1\cdot E(M)\cdot E(N|V=v)$$

$$= \{\rho\cdot(q2/2q1 - q1) + q1\cdot E(M)\} + q1\cdot E(M)\cdot(1-\ell)\cdot v/w1, \qquad v \geq 0$$

Expected total response conditioned on v is the sum of the wait in the queue, mean overhead associated with this task, $(w1 + (1-\ell)\cdot v)\cdot d1/w1$, and the service time of the task v/c.

$$(2.49)\quad E(R|V=v) = E(T|V=v) + (w1 + (1-\ell)\cdot v)\cdot d1/w1 + v/C$$

$$= \alpha + B\cdot v$$

$$\text{where}\quad \alpha = \rho\cdot(q2/2q1 - q1) + q1\cdot E(M) + d1$$

$$B = 1/C + (q1\cdot E(M) + d1)\cdot(1-\ell)/w1$$

The term α of the previous equation is the expected value of the minimum response time possible in the system. This unavoidable delay is the sum of the task's overhead time and the processing and overhead times of the jobs already in the queue. A physical interpretation of this term is the response time to a null input (e.g., a carriage return). Note that after this initial delay, expected response is a linear function of service request v. A common aspect of many current time sharing models is an essentially linear relationship between response and service request. This characteristic is present in the earliest models as shown by the form of equations (1.2) and (1.3) derived by Kleinrock, but these early models do not include important features such as random quanta and random overhead. E(R) may be obtained from (2.49) by removing the condition on v. Since the expression is linear, one simply replaces v with E(V).

2.4 THE TANDEM QUEUEING MODEL - TSMOD2

A characterisitc shared by almost all computer programs is that they can be represented as repeating cycles of central processor activity followed by utilization of the input-output, (I/O), system. Multiprogramming designs allow different programs to use these facilities simultaneously by

switching control from a program requesting I/O service to one needing the central processing unit. When the original program has finished I/O activity it may queue for additional central processing time and release the I/O facility. Time-sharing operating systems often force task switching by making a program release control of the central processor when it has exceeded a quantum processing limit. Figure 2.5 illustrates a basic tandem, two server model of this organization. For analytical purposes it clearly does not matter which server is considered the central processor and which the I/O system.

Define the following symbols for use in the model.

ℓ = the probability that a job leaves the system after a cycle of processing and I/O activity

C_i = a constant equal to the processing rate of subsystem i expressed in work per unit time

W_i = the exponentially distributed random work required from subsystem i during a processing cycle. The expected value of W_i is $w1_i$.

$\mu_i = C_i/w1_i$ = the exponential service rate of subsystem i

N = the random number of cycles required by a task to finish one complete interaction with the system

V = an exponentially distributed random variable denoting the total work required by a task from the central processor in N cycles. The expected value of V is $w1_1/\ell$.

M_i = the random number of tasks waiting in queue i, and being served in subsystem i

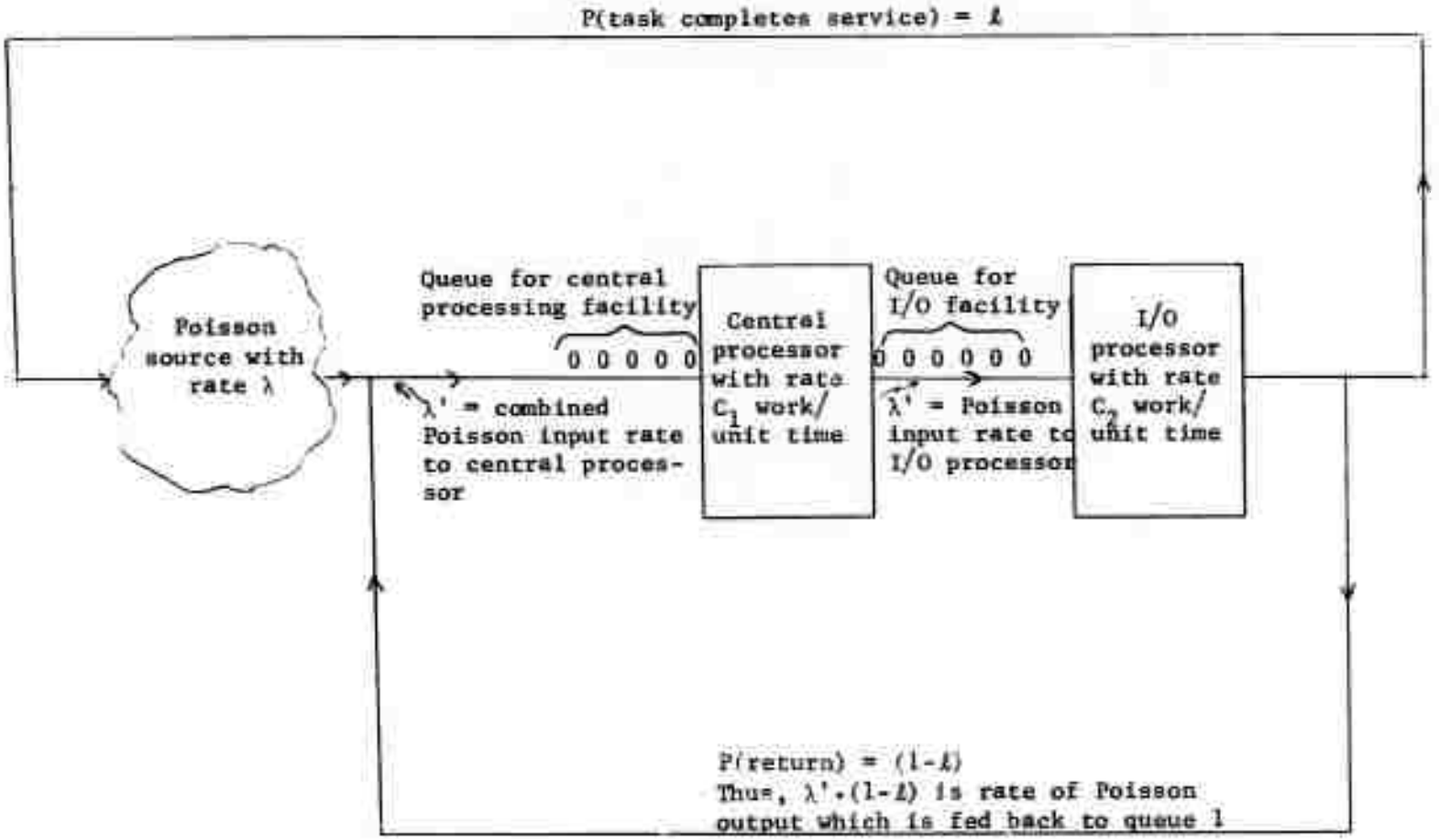


Figure 2.5

Structure of Tandem Queueing System - TSMOD2

The assumptions for the model are:

(a) the input process is Poisson with rate λ

(b) queue 1 and queue 2 have unlimited capacity

(c) the service time in each processor is exponential with mean wl_1/C_1 and wl_2/C_2, respectively

(d) after completing service in the second processor a task rejoins queue 1 with probability $(1-\ell)$ and leaves the system with probability (ℓ). The probability of rejoining queue 1 is independent of all other state variables.

Jackson (1963) presents a number of important results for networks of Poisson queues. A summary of many of his derivations appears in Conway, Maxwell, and Miller (1967), Chapter 10. Call the combined input rate to the first processor λ'. It is the sum of the external Poisson input, of rate λ, and that portion of the I/O system's output which is fed back to the first queue. This latter process has a rate of $\lambda' \cdot (1-\ell)$. Thus λ', the rate of the combined input, is:

$$\lambda' = \lambda + (1-\ell) \cdot \lambda' \tag{2.50}$$

$$\text{or } \lambda' = \lambda/\ell$$

A key result of Jackson's analysis is that in the network illustrated in Figure 2.5 the combined input processes and the resulting output processes are all Poisson. Thus each subsystem may be analyzed as an exponential server having Poisson input.

In the steady state each of the two servers may be treated as an independent M/M/1 queue with input rate λ', and service rate $\mu_i = C_i/wl_i$. Equation (2.10) presents expected response in an M/G/1 queueing system. This equation reduces significantly when service is exponential. Thus expected response through subsystem i, and the sum of the expected number of tasks waiting in queue and being served in subsystem i, $E(M_i)$, are:

$$(2.51)\quad E(R_i) = 1/(\mu_i - \lambda') \qquad i=1,2,\ \mu_i > \lambda'$$

$$(2.52)\quad E(M_i) = \lambda' \cdot E(R_i) = \lambda'/(\mu_i - \lambda'), \qquad i=1,2,\ \mu_i > \lambda'$$

Let T_i be the time spent waiting for the central processor plus the time waiting and receiving service from the I/O processor on the i^{th} pass through the system. Since all of the stochastic subsystems in this model are Poisson, in steady state, job arrivals and departures occur at random points in time. For exponential service, the remaining processing time of a task is also exponential regardless of how much service the task has already received. Thus the wait in queue 1 is $E(M_1) \cdot E(\text{service quantum})$, and the mean response time through system 2 is $1/(\mu_2 - \lambda')$. In steady state all cycle times have the same expected value.

$$(2.53)\quad E(T_i) = E(\text{wait for processor}) + E(\text{wait + service for I/O system})$$

$$= \frac{\lambda'}{\mu_1(\mu_1 - \lambda')} + \frac{1}{(\mu_2 - \lambda')}, \qquad \mu_1 \text{ and } \mu_2 > \lambda' \quad i=1,2,\ldots$$

The conditional response time of a job requiring v units from the central processor which it receives in n quanta is:

$$(2.54)\quad E(R|V=v,N=n) = n\cdot\left\{\frac{\lambda'}{\mu_1(\mu_1-\lambda')} + \frac{1}{(\mu_2-\lambda')}\right\} + v/C_1, \quad n=1,2,3\ldots$$

$$v \geq 0$$

$$\mu_1 \text{ and } \mu_2 > \lambda'$$

Removing the condition by using (2.46) leads to the result for expected response conditioned on service request.

$$(2.55)\quad E(R|V=v) = \alpha + B\cdot v/C_1$$

$$\text{where}\quad \alpha = \frac{\lambda'}{\mu_1(\mu_1-\lambda')} + \frac{1}{(\mu_2-\lambda')}$$

$$\text{and}\quad B = \mu_1\cdot(1-\ell)\cdot\alpha + 1$$

Figure 2.6 displays the non-linear effect of increasing the demand on the system. For each line on this graph, the processing request, v, is held constant and the arrival rate, λ, is increased. Figure 2.7 is a graph of expected response conditioned on service request, v, for a number of input rates λ. Thus this model also predicts both a linear relationship between expected response and service request, and non-linear response degradation as a function of system load.

2.5 A PROCESSOR-SHARED MODEL WITH STATE DEPENDENT OVERHEAD, ARRIVAL, AND SERVICE PROCESSES - TSMOD3

A fundamental concept of time-sharing organizations is that the power of the central processing unit is to be allocated to all tasks demanding service. Processor-shared models approximate actual scheduling procedures, such as round robin time slicing, with an ideal discipline in which fixed processor capacity, C, is divided uniformly and delivered to all active tasks. At every instant, each of m active jobs receives C/m

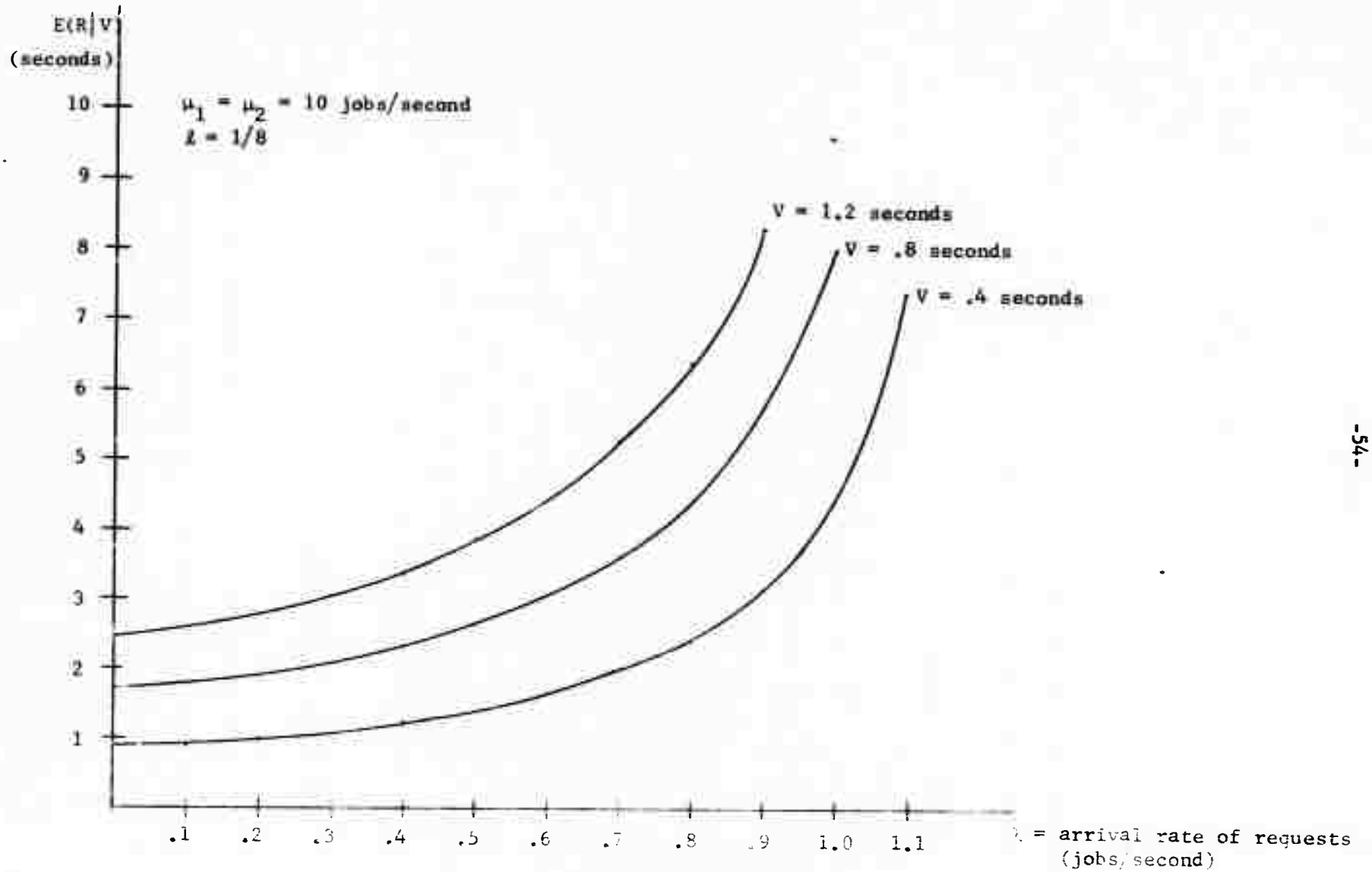


Figure 2.6

Expected Response as a Function of System Load - TSMOD2

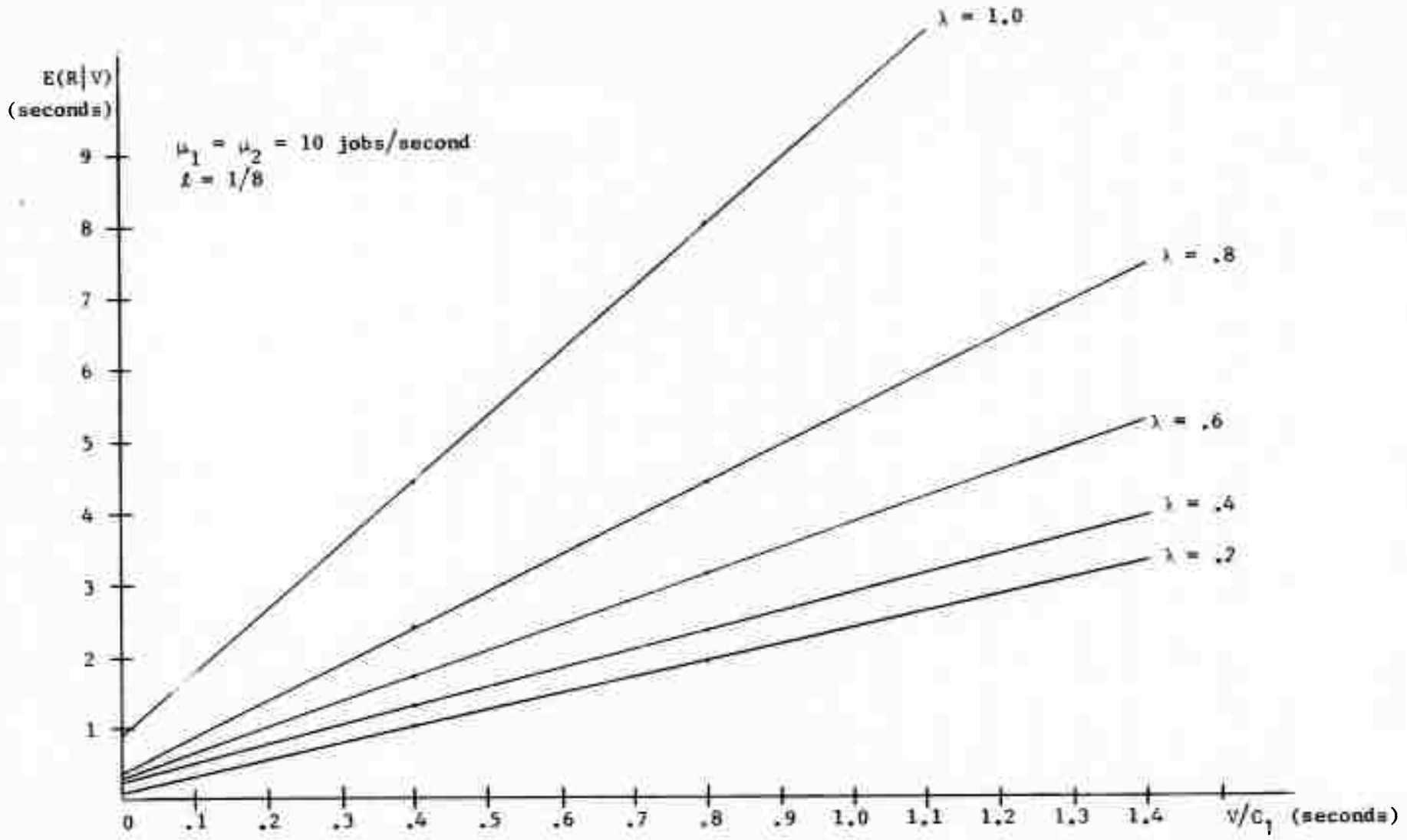


Figure 2.7

Expected Response as a Function of Service Request V - TSMOD2

units of computing power per unit time. Scherr (1967) recognized that one of the classic forms of the general "birth and death" model was directly applicable to the time-sharing problem. His formulation of the problem allows one to consider explicitly the number of terminals connected to the system. Scherr considered overhead in a simplified manner by reducing the capacity of the central processor from C to a lesser value C'. The quantity (1-C'/C) represents the fraction of the capacity lost to overhead.

Van de Goor (1970) measured a number of overhead factors in a small time-sharing system. He discovered that a significant portion of overhead is proportional to the number of active tasks demanding service from the system. For example, both paging activity in a virtual memory organization and many monitor list searching operations are proportional to the number of active tasks. The mathematical structure of a finite source, processor-shared model, allows one to incorporate overhead loss that is proportional to the number of active tasks in the system. If there are m active jobs demanding service, then at each instant every task will seem to have its own virtual processor with capacity $(1-f \cdot m) \cdot C/m$ instructions per unit time. To keep capacity positive the overhead loss fraction, f, must be less than $1/N$ where N is the number of terminals connected to the system.

Each of the N input terminals is an exponential source with rate λ. However, once a terminal has submitted a job, it is blocked from additional input activity until the computer completes its request. The combined total input rate for all terminals that do not have requests pending is $(N-m) \cdot \lambda$, where m is the number of jobs actively using the processor. All

service requests are drawn from an exponential distribution with parameter v. Thus the mean service request is $1/v$ instructions, and the rate at which the server processes jobs is $v \cdot (1-f \cdot m) \cdot C$. An important feature of this type of model is that it is stable in the sense that the input rate decreases as the number of tasks demanding service increases. Unlike the models in the previous sections where the queues could become unbounded, this structure is self correcting, and a steady state solution will always exist. Figure 2.8 illustrates the basic organization of the model.

The standard method of solving this class of model is to form a set of differential difference equations involving system state variables. Let $P_m(t)$ be the probability that there are m active tasks in the processor at time t. Since all of the individual input and service processes are exponential this continuous-time Markov model has simple state transition probabilities. For example, when $1 < m < N$ the general state equation is:

$$
\begin{aligned}
(2.56)\quad P_m(t+\delta) = {} & (N-m+1) \cdot \lambda \cdot \delta \cdot (1-f \cdot (m-1)) \cdot v \cdot C \cdot \delta) \cdot P_{m-1}(t) \\
& + (1-(N-m) \cdot \lambda \cdot \delta) \cdot (1-(1-f \cdot m) \cdot v \cdot C \cdot \delta) \cdot P_m(t) \\
& + (1-f \cdot (m+1)) \cdot v \cdot C \cdot \delta \cdot (1-(N-m-1) \cdot \lambda \cdot \delta) \cdot P_{m+1}(t) + o(\delta^2)
\end{aligned}
$$

The basic principles underlying this equation are that if interevent times have an exponential distribution with rate y, then:

(a) the probability that an event will occur during an interval of length δ is $y \cdot \delta$, and the probability of no event occurring is $(1-y \cdot \delta)$

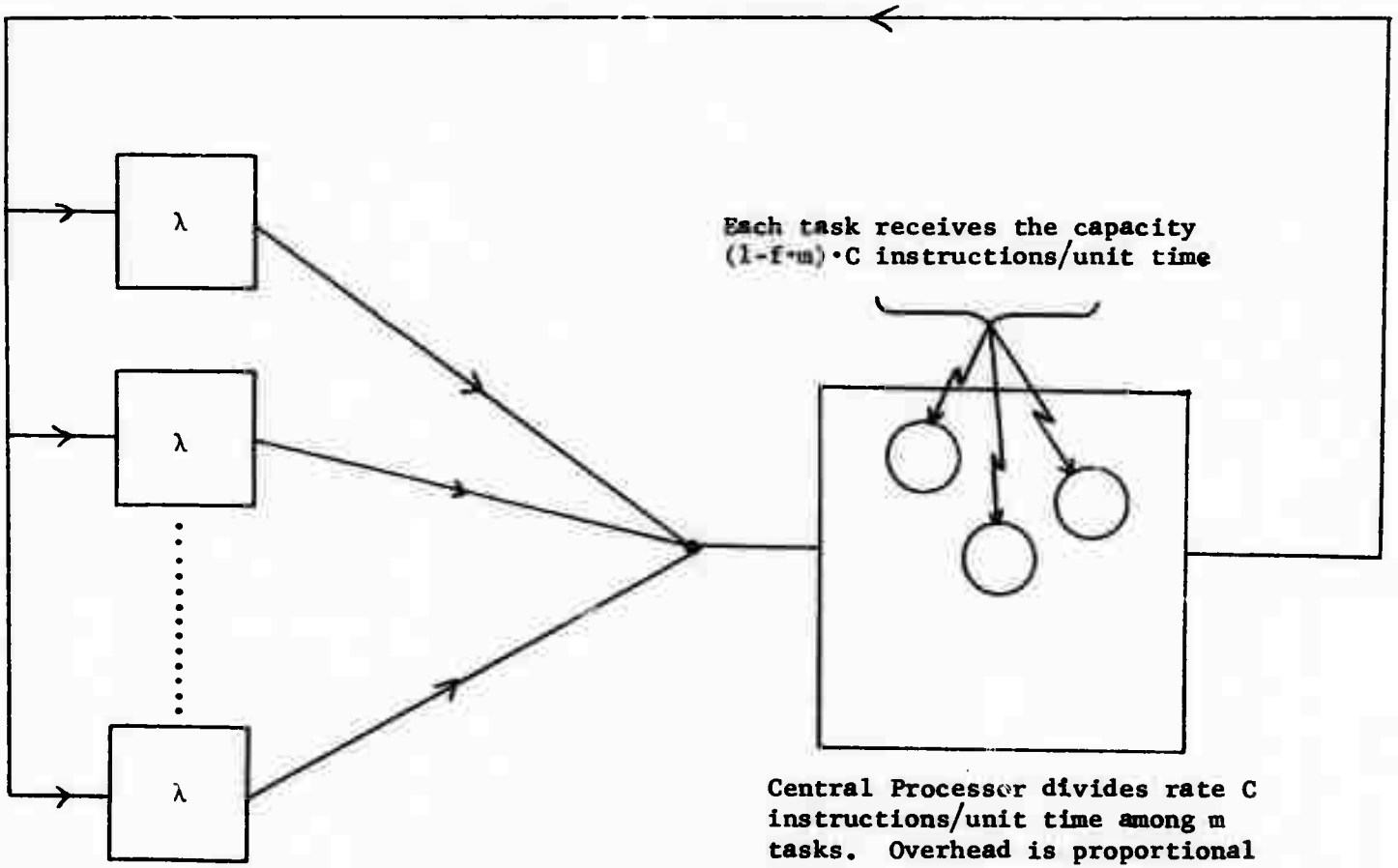


Figure 2.8
Structure of Finite Source, Processor Shared Model - TSMOD3

(b) if there are k such processes working in parallel, then the probability that an event will occur during an interval of length δ is $k \cdot y \cdot \delta$

(c) the probability of two or more events occurring during δ is of the order of $\delta \cdot \delta$, i.e., $o(\delta^2)$.

The next steps in the derivation are to construct similar equations for the two boundary states m=0 and m=N, and take the limit as $\delta \to 0$. Let the derivative of the state probability with respect to time be $P'_m(t)$

$$(2.57) \quad \text{limit}_{\delta \to 0} \frac{P_m(t+\delta) - P_m(t)}{\delta} = P'_m(t)$$

The set of differential difference state equations becomes:

$$(2.58) \quad P'_0(t) = -N \cdot \lambda \cdot P_0(t) + (1-f) \cdot v \cdot C \cdot P_1(t)$$

$$\vdots$$

$$P'_m(t) = (N-(m-1)) \cdot \lambda \cdot P_{m-1}(t) - \{(N-m) \cdot \lambda + (1-m \cdot f) v \cdot C\} \cdot P_m(t)$$

$$+ (1-(m+1) \cdot f) \cdot v \cdot C \cdot P_{m+1}(t) \quad , \quad m=1,2,\ldots N-1$$

$$\vdots$$

$$P'_N(t) = \lambda \cdot P_{N-1}(t) - \{(N-N) \cdot \lambda + (1-N \cdot f) \cdot v \cdot C\} \cdot P_N(t)$$

Statistical equilibrium (or steady state) exists when the state probabilities no longer change with time.

$$(2.59) \quad \lim_{t \to \infty} P'_m(t) = 0$$

$$(2.60) \quad \lim_{t \to \infty} P_m(t) = P_m$$

To solve for the equilibrium state probabilities, one lets all $P'_m(t) = 0$ and then uses the resulting recursive set of steady state equations, and the fact that the sum of all of the probabilities is unity, to compute all values of P_m. Setting all $P'_m(t)$ to zero and reworking equation (2.58) by substituting the result for P_m into the equation for P_{m+1} leads to the following set of steady state equations.

$$(2.61)\quad \begin{aligned} (1-m\cdot f)\cdot v\cdot C\cdot P_m &= (N-(m-1))\cdot\lambda\cdot P_{m-1}, \quad m=1,2,\ldots N,\ f < 1/N \\ &\vdots \\ 0 &= (N-N)\cdot\lambda\cdot P_N \end{aligned}$$

Adding all terms on both sides of this set of equations produces the following expression.

$$(2.62)\quad v\cdot C\cdot \sum_{m=1}^{N} P_m - v\cdot C\cdot f\cdot \sum_{m=0}^{N} m\cdot P_m = N\cdot\lambda\cdot \sum_{m=0}^{N} P_m - \lambda\cdot \sum_{m=0}^{N} mP_m$$

Substituting equations (2.63) and (2.64) into (2.62) leads to (2.65), the result for the expected value of the number of tasks demanding service from the system, E(M).

$$(2.63)\quad E(M) = \sum_{m=0}^{N} m\cdot P_m$$

$$(2.64)\quad \sum_{m=1}^{N} P_m = 1 - P_0$$

$$(2.65)\quad E(M) = \{N\cdot\lambda - v\cdot C\cdot(1-P_0)\}/(\lambda-(v\cdot C)\cdot f),\ f < 1/N,\ f \neq \frac{\lambda}{v\cdot C}$$

Equation (2.64) and the set of equations (2.61) lead to the derivation of P_0, the probability that the central processor is idle. All of the

other state probabilities are expressible in terms of P_0.

$$(2.66)\quad P_1 = N\cdot\lambda\cdot P_0/\{v\cdot C\cdot(1-f)\}$$

$$P_2 = N\cdot(N-1)\cdot\lambda^2\cdot P_0/\{(v\cdot C)^2\cdot(1-f)(1-2f)\}$$

$$\vdots$$

$$P_m = P_0 \prod_{i=1}^{m} \{(N-(i-1))\cdot\lambda/(v\cdot C\cdot(1-i\cdot f))\},\quad m=1,2,\ldots,N$$

where $\prod_{i=1}^{m} X_i = X_1\cdot X_2\cdot X_3\ldots X_m$

$$(2.67)\quad P_0 = 1/\left[1 + \sum_{m=1}^{N}\prod_{i=1}^{m} \{(N-(i-1))\cdot\lambda/(v\cdot C\cdot(1-i\cdot f))\}\right]$$

If the state dependent overhead fraction, f, is zero, then the result reduces to the classic formula for the exponential machine repair problem.[4]

To express mean response time as a function of the mean number in the system (equation 2.65) one may use the equilibrium argument that the mean number of jobs submitted to the system per unit time must equal the mean number served per unit time. Each of the N terminals goes through many cycles of generating a request and then waiting for the system to respond to that request. The mean time spent in the first part of this cycle is $1/\lambda$ time units and the mean time spent in the second is E(R) time units. Thus the mean arrival rate from each terminal is $1/(1/\lambda+E(R))$ and the total mean arrival rate to the system is N times this quantity. The service rate of the system is $v\cdot C\cdot(1-m\cdot f)$ where m is the number of tasks being served.

---

[4] See Saaty (1961), p. 326.

$$(2.68)\quad N/\left(1/\lambda+E(R)\right) = 0\cdot P_0 + vC\cdot\sum_{m=1}^{N}(1-m\cdot f)\cdot P_m$$

$$= v\cdot C\cdot(1-P_0) - v\cdot C\cdot f\cdot E(M)$$

Another way of looking at this relation is to note that when there are m tasks in the system the arrival rate from the remaining terminals is $(N-m)\cdot\lambda$ and thus the mean arrival rate is $(N-E(M))\cdot\lambda$. One may equate both expressions for mean arrival rate.

$$(2.69)\quad N/\left(1/\lambda+E(R)\right) = (N-E(M))\cdot\lambda$$

By using equation (2.65) in (2.68), or more simply by solving (2.69) for E(R), one may obtain the following result for mean response time in TSMOD3. Both approaches lead to the same result.

$$(2.70)\quad E(R) = \frac{E(M)}{\lambda\cdot(N-E(M))}$$

Figure 2.9 illustrates mean response as a function of N, the number of terminals connected to the processor, for a number of values of f, the overhead loss function.

Each of the models developed in this chapter focusses on a different aspect of current implementations of time-shared computing systems. The inherent complexities of queueing models make the simultaneous consideration of all such features very difficult. The next two chapters present empirical investigations of both simulated and actual systems. Response time measures of these more complex systems are compared with the predictions of the analytic models of this chapter. Chapter 5 contains a number of examples of how one may use these models.

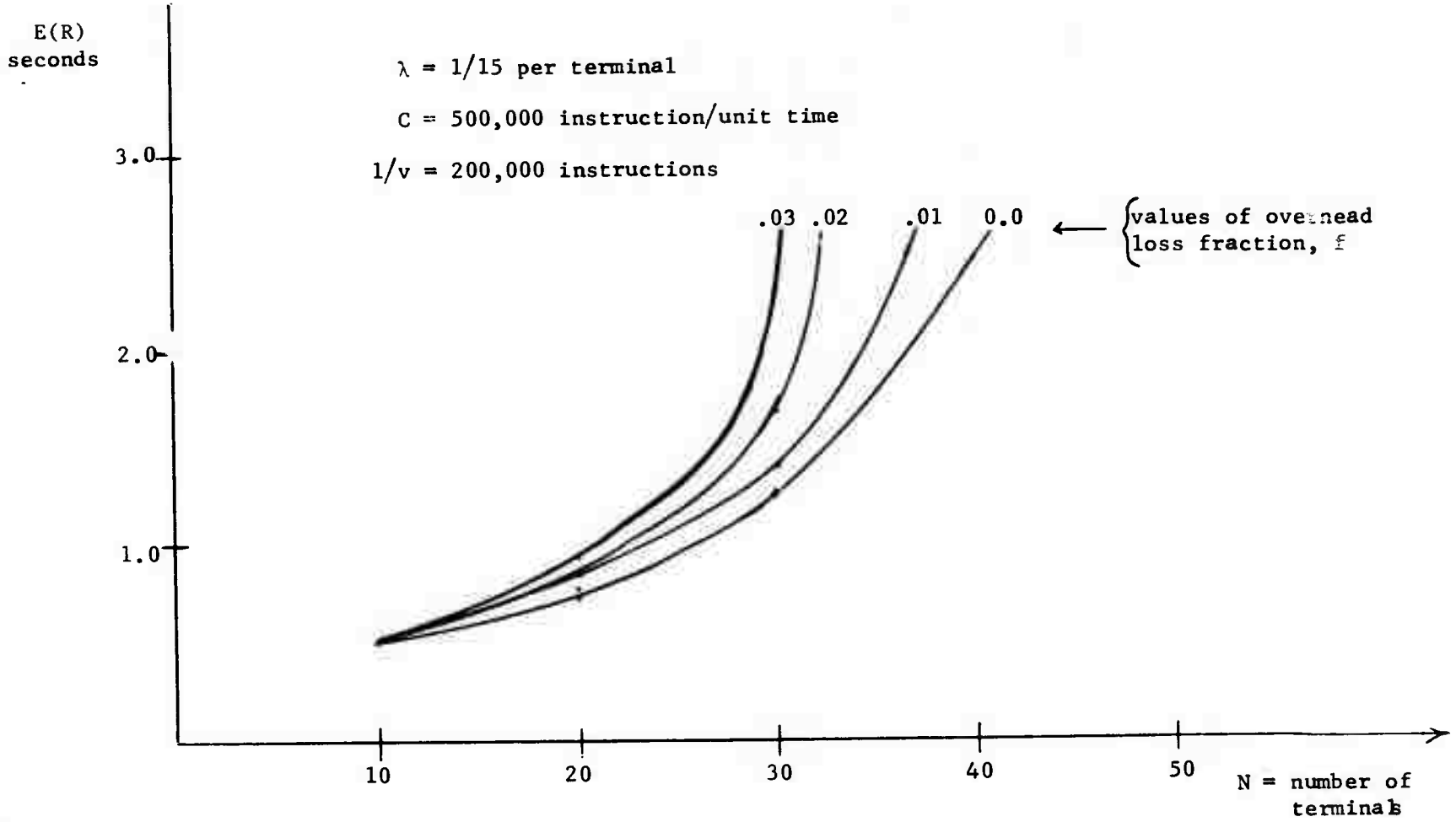


Figure 2.9

Mean Response as a Function of the Number of Terminals - TSMOD3

CHAPTER 3

SIMULATION STUDIES OF SYSTEM BEHAVIOR

3.1 INTRODUCTION

The results of Chapter 2 provide new expressions relating response time measures of system performance to parameters such as overhead loss, processing capacity, service and arrival distributions, and interrupt probabilities. To keep results easy to compute, these analytic models are based on many simplifying assumptions concerning system architecture and user behavior. In addition, equation (2.49), the least complicated expression for mean response conditioned on service request, depends on the approximation that all cycle times after the first are equal to the mean number of tasks in the system multiplied by the mean quantum interval. The goal of this chapter is to explore the robustness of these results when they are applied to systems that do not satisfy all of the assumptions. The following experiments range from simulations closely related to the analytic models of Chapter 2 to more complex designs based on features of an operational time-shared system.

The first simulation is an exact model of Figure 2.1, with overhead and quantum times both having truncated normal distributions (both were constrained to be non-negative). The second and third models are based on a tandem queueing structure like that analyzed in Section 2.4. The last simulation in the chapter includes a detailed model of the scheduling algorithms of TSS, an operating system for the IBM 360/67. Task dispatching in this system includes dynamic priorities, and is much more complex than the cyclic, round robin, scheduling of the previous models.

The goals of the experiments are:

(a) to determine how well the equations of Chapter 2 predict performance characteristics, such as expected value of response conditioned on service request, even though the models differ from the assumptions underlying the previous derivations

(b) to study performance characteristics that were not derived analytically but are easy to examine by simulation and which lead to a deeper understanding of feedback queueing systems

(c) to determine if a complex model based on an operational system exhibits the same basic characteristics as the simpler models.

3.2 EXPERIMENTAL METHODOLOGY

3.2.1 The Simulations

All of the models are implemented in SIMULA, a general purpose simulation language which extends ALGOL in a number of important dimensions. In addition to all of the features of ALGOL, the language provides good list processing capabilities, a powerful co-routine capability including a full range of process scheduling mechanisms, and a number of statistical procedures.[1]

For the first three studies each experiment consisted of a 100 task initialization period, in which statistics were not gathered, followed

[1] The reader interested in SIMULA is directed to Dahl and Nygaard (1966), Univac (1967), and McCredie (1970).

by a production period in which statistics were calculated for 1000 tasks passing through the system. Pilot runs produced initial estimates for running times and variances of the sample statistics. Each experimental run of the first study required approximately 10 seconds of Univac 1108 processing time. Since the second and third models have two processing subsystems in tandem they required twice as much computer time per simulation as the initial model. To simulate the processing of 1000 tasks in the complex model described in Section 3.5 required about three or four minutes of 1108 time. As a result of the expense associated with the detail of this model, only a few experiments were performed.

Each run represents an independent set of statistics since the models were initialized with different starting seeds for random number generators, and all statistical counters were reset to zero. The initialization period to remove startup transients preceded each run. Appendix B contains listings of the simulations used for the studies.

3.2.2 The Statistical Analysis

One must use statistical tools to analyze data from stochastic systems. A striking characteristic of the data from the simple queueing structures of Sections 3.3 and 3.4 is its high variance. The estimators used to determine model variables come from independent experiments. Label the value of an estimator from simulation run i, X_i. Each independent X_i is drawn from a population having a finite mean μ and variance σ^2. An estimator of μ is the sample average $\bar{X}$, which is based on all of the experiments and is itself a random variable. In Section 3.3 and 3.4 each study consists of 20 independent experiments ($n = 20$).

$$(3.1)\quad \bar{X} = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i$$

The population variance, σ^2, for each variable, is unknown in the experiment, but one may use the following estimator of it.

$$(3.2)\quad s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2$$

To scale $\bar{X}$ so that it has a mean of zero and variance of unity, subtract μ from $\bar{X}$ and divide the result by $(s^2/n)^{\frac{1}{2}}$.

$$(3.3)\quad Z = (\bar{X}-\mu)/(s^2/n)^{\frac{1}{2}} = (n)^{\frac{1}{2}}(\bar{X}-\mu)/s$$

The central limit theorem states that Z becomes normally distributed with mean 0 and variance 1 as n becomes large. Z does not have a normal distribution for small n because it is based on the random variable s^2, an estimate of σ^2. Z has a Student-t distribution which deviates from the Normal distribution for small values of n, but approaches the Normal when n is large (e.g., $n > 30$).[2]

One may form a confidence interval for sample averages by locating points which partition a desired percent of the area under the density function for the Student-t distribution. For example, one may compute the probability that an interval based on sample statistics covers the true mean, μ. Using the Student-t distribution with n-1 = 19 degress of freedom, one finds that in these experiments the probability is .95 that the interval of equation (3.4) will contain the true mean, μ.

---

[2] See Mood and Graybill (1963), pp. 251-253 for a discussion of the estimation of mean values when the variance is not known and pp. 149-153 for a discussion of the central limit theorem.

$$(3.4)\quad \bar{X} - 2.09\cdot(s^2/n)^{\frac{1}{2}} \le \mu \le \bar{X} + 2.09\cdot(s^2/n)^{\frac{1}{2}}$$

3.3. A SIMULATION OF TSMOD1

3.3.1 The Model

TSMOD1, the feedback queueing model studied in Section 2.3 and illustrated in Figure 2.1, is the subject of the first validation experiment. The following parameters were used in the study:

$F_W(t) = F_D(t) \sim$ normal distributions (mean=.05, σ=.015)

$w1/C = d1 = \mu = .05$ seconds

$w2/C^2 = d2 = \sigma^2+(\mu)^2 = .002725(\text{seconds})^2$

$w3/C^3 = d3 = (\mu)^3+3\cdot\mu\cdot\sigma^2 = .00015875(\text{seconds})^3$

Processing rate = C = 1 instruction/microsecond

Arrival rate = λ = 1 job/second

Probability job leaves after an interrupt = $\ell = 1/8$

The statistical estimators used to summarize the data are:

(3.5) $\bar{R}$ = the sample average of response times

$$= \frac{1}{n}\cdot\sum_{i=1}^{n} R_i$$

where R_i = (time task i leaves the system - time task i entered the system)

(3.6) SD(R) = the sample standard deviation of response times

$$= \left\{\frac{1}{n-1}\cdot\sum_{i=1}^{n} (R_i-\bar{R})^2\right\}^{\frac{1}{2}}$$

(3.7) $\bar{t}_1$ = the sample average of the time spent waiting until a task receives its first processing quantum

$$= \frac{1}{n} \cdot \sum_{i=1}^{n} \text{(time task i waits before beginning its first processing quantum)}$$

(3.8) $\overline{P0} \cdot 100$ = the percentage of time the processor was idle

$$= 100 \cdot \text{(time processor idle/total simulated time)}$$

(3.9) $\bar{B}$ = least squares estimate of slope of response time as a function of service request (where v_i is service request of the i^{th} task)

$$= \{n \cdot \sum_{i=1}^{n} R_i \cdot v_i - \sum_{i=1}^{n} R_i \cdot \sum_{i=1}^{n} v_i\} / \{n \cdot \sum_{i=1}^{n} v_i^2 - (\sum_{i=1}^{n} v_i)^2\}$$

The analysis of Section 2.3 presents exact solutions for the expected values of $\bar{R}$ (equation 2.24), SD(R) (equation 2.19), $\bar{t}_1$ (equation 2.31), and $\overline{P0}$ (equations 2.10 and 2.16). Equation (2.49) is an approximate expression for the expected value of response time conditioned on service request. This equation contains a parameter B which is the slope of the conditional response time. The approximation is based on the assumption that the service request, V, is an exponentially distributed random variable, and on the mathematical simplification that all cycle times after the first are equal to the mean number of tasks in the system multiplied by the mean quantum interval. The next section contains comparisons of the results of the simulations of TSMOD1 with the analytic expressions for these variables.

3.3.2 The Results

Table 3.2 presents all of the experiemtnal results for each of the five variables for 20 independent runs, each of which represents 1000 observations obtained after an initialization period of 100 tasks. The experimental values are combined to form the estimators $\bar{X}$, s^2, s, $(s^2/20)^{\frac{1}{2}}$ defined by equations (3.1) and (3.2). Table 3.1 summarizes this data by presenting the 95 percent confidence interval from the experimental data and the analytic result from Chapter 2 for each of the five variables. All of the analytic results lie within the confidence intervals. The samples display the high variance inherent in queueing systems of this type. This particular sample exhibits slightly heavier congestion than predicted by the analytic solution. For example, runs six and ten are very heavily congested experiments. Figure 3.1 is a typical histogram of response times in TSMOD1.

Variable	95 percent confidence interval [see eq. (3.4)]	Sample Average	Analytic Result
$\bar{R}$	$3.35 \leq E(R) \leq 4.51$	3.93	3.81
SD(R)	$4.47 \leq SD(R) \leq 6.31$	5.39	5.58
$\overline{P0} \cdot 100$	$17.38 \leq p_0 \cdot 100 \leq 20.43$	18.90	20.00
$\overline{t_1}$	$.34 \leq E(t_1) \leq .47$	.41	.39
$\bar{B}$	$8.33 \leq E(B) \leq 11.28$	9.81	8.54

TABLE 3.1

Comparison of Experimental and Analytic Values for TSMOD1

Run No.	$\bar{R}$	SD(R)	$\bar{t}_1$	$\bar{P}_0 \cdot 100$	$\bar{B}$
1	3,43	4.87	.35	22.53	8.96
2	3.21	4.09	.32	20.80	7.71
3	4.10	5.78	.41	18.33	10.93
4	3.30	4.25	.32	22.10	8.25
5	3.50	5.35	.36	20.44	8.20
6	7.38	10.92	.79	14.21	19.79
7	3.22	4.22	.32	22.32	7.81
8	3.69	5.39	.39	18.36	9.53
9	3.59	4.33	.37	14.53	9.05
10	6.73	8.97	.72	12.26	15.46
11	4.29	5.57	.46	14.14	10.20
12	2.92	4.43	.29	22.62	8.01
13	4.37	5.94	.47	17.87	10.24
14	3.09	4.15	.32	21.08	7.93
15	4.89	6.20	.52	14.97	11.65
16	2.73	3.15	.28	20.16	6.91
17	2.86	3.49	.28	21.63	6.59
18	3.79	5.54	.38	21.30	10.17
19	4.68	8.03	.48	18.12	11.80
20	2.84	3.23	.28	20.32	7.00
$\bar{X}$	3.93	5.39	.41	18.90	9.81
s^2	1.53	3.84	.02	10.64	9.94
s	1.24	1.96	.14	3.26	3.15
$(s^2/20)^{\frac{1}{2}}$	.28	.44	.03	.73	.70

Table 3.2

Simulation Results for Experiments on TSMOD1
(Input parameters are displayed in Section 3.3.1)

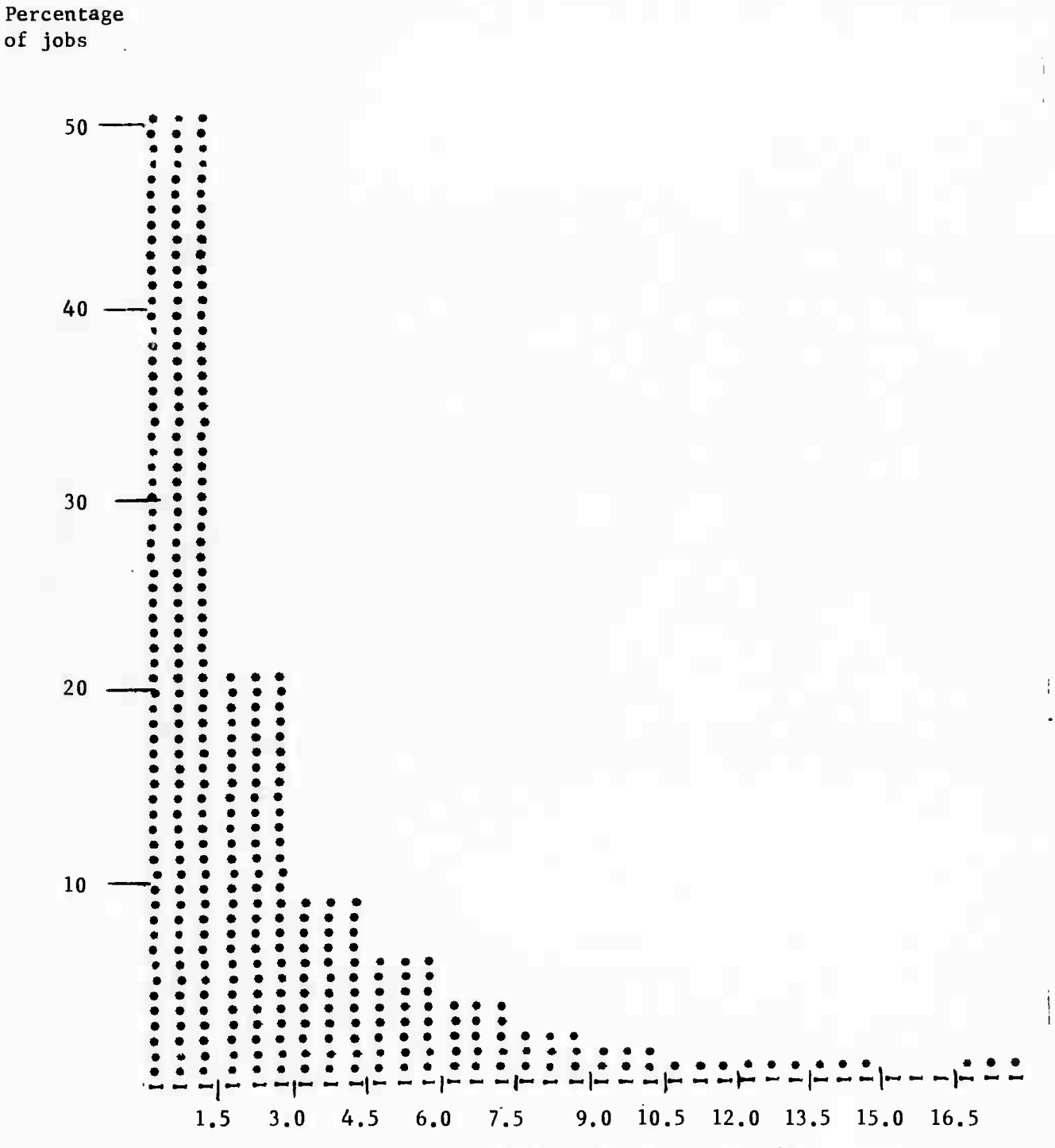


Figure 3.1

Histogram of Response Times for TSMOD1

The analysis of Section 2.3 predicts a nearly linear relationship between service request and mean response. To test this result, each task recorded both its service request and its transit time through the system. These data were separated into equal service intervals (and an additional overflow interval) and processed to calculate mean job response for each interval. Table 3.3 displays the average response, and the number of observations on which it is based, for five service intervals for the 20 experimental runs. Results for the $\bar{X}$, s^2, s, and $(s^2/n)^{\frac{1}{2}}$ are also displayed.

Figure 3.2 is a graph of the data of Table 3.3. Each point is the sample average of the observations in that interval, and is placed at the mid point of the service interval. The vertical bars through each point represent the sizes of the 95 percent confidence intervals computed using equation (3.4). The solid line connects the sample averages and the two broken lines form an area in which the true value of mean response conditioned on service request lies with 95 percent confidence. The increasing variance with service comes from two factors: (a) the number of observations decreases in the intervals having higher values of service and (b) variance of response increases with service request. Thus long jobs experience longer, and more highly variable, response than short jobs.

Experiment	.15 ≤ v < .2		.35 ≤ v < .4		.55 ≤ v < .6		.75 ≤ v < .8		.95 ≤ v < 1.0	
1	1.12	105	3.38	50	5.12	33	6.73	24	10.44	12
2	1.07	96	2.64	53	3.55	31	5.44	12	7.74	11
3	1.55	104	2.54	53	5.06	38	8.90	22	7.07	8
4	1.05	94	3.18	57	4.05	34	5.46	20	7.00	13
5	1.43	97	3.15	50	4.09	26	5.40	17	4.70	12
6	2.72	99	6.12	55	9.98	43	13.16	29	15.88	4
7	1.13	96	2.89	47	3.19	24	6.50	19	8.54	13
8	1.56	90	3.41	57	3.68	38	8.11	17	8.14	9
9	1.40	93	2.92	63	5.65	38	6.48	13	7.90	11
10	2.38	76	6.10	59	10.34	24	10.76	23	12.88	18
11	1.71	76	3.99	71	5.27	30	8.14	28	6.56	11
12	1.16	91	2.55	56	4.10	33	6.10	18	8.25	8
13	1.51	85	4.27	59	6.39	44	6.73	23	12.00	9
14	1.23	93	3.29	40	3.99	31	6.31	15	7.50	13
15	1.95	96	4.16	56	6.92	29	9.69	20	11.09	12
16	1.15	91	2.34	59	3.60	36	5.87	21	6.87	8
17	1.02	102	2.35	59	4.01	31	5.22	22	7.59	18
18	1.31	90	3.21	55	4.63	28	5.09	17	8.90	12
19	1.82	107	3.67	45	5.71	21	8.31	17	18.76	2
20	1.08	83	2.48	56	3.52	30	5.56	22	7.30	19
$\bar{X}$	1.47		3.43		5.15		7.20		9.26	
s^2	.21		1.17		3.97		4.52		11.58	
s	.46		1.08		1.99		2.13		3.40	
$(s^2/20)^{\frac{1}{2}}$	.10		.24		.45		.48		.76	

Table 3.3

Average Response Time as a Function of Service Request - TSMOD1

Note: Within each interval of service request (v, measured in seconds), the first column contains the experimental average of response times for jobs having requests in the interval, and the second column contains the number of such jobs for each experiment.

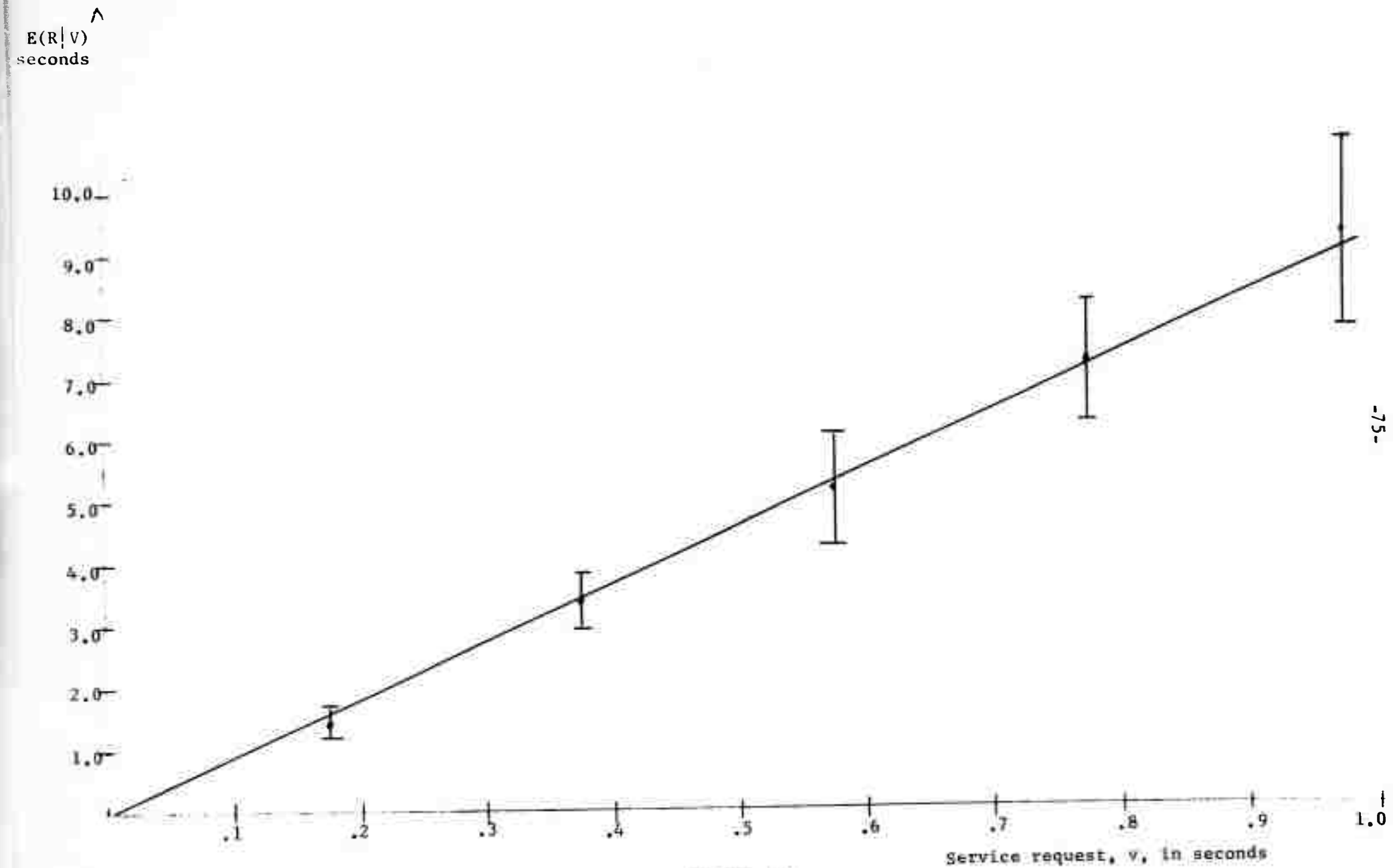


Figure 3.2

Expected Response Conditioned on Service Request - TSMODl

3.4 A SIMULATION OF TSMOD2

3.4.1 The Models

TSMOD2 is similar to TSMOD1 with an additional processing system after the central processing unit. This extra queueing subsystem represents an input/lutput (I/O) system for program swapping, paging, and file handling. Figure 2.5 illustrates the structure and Section 2.4 includes an analysis of the model when the service times in both systems are exponential. This section includes simulations of two versions of TSMOD2. In the first (which is identical to the model of Section 2.4) both the processor and the I/O systems have exponential service distributions and there is no explicit overhead delay. In the second, the processor is identical to TSMOD1 (an overhead delay and then a processing quantum) and the I/O processor has a uniform service distribution which is more representative of rotating external storage devices than an exponential distribution. The following parameters were used for these two models:

	Version I	Version II
Server 1	Exponential distribution with rate 10 jobs/second. No overhead.	Overhead and service quanta both have Normal distributions with mean = .05 seconds and standard deviations = .015 seconds (rate = 10 jobs/second)
Server 2	Exponential distribution with rate 10 jobs/second.	Uniform distribution between 0.0 and .2 seconds (rate = 10 jobs/second).
System arrival rate= λ	1 job/second	1 job/second
Probability job leaves after an interrupt = ℓ	$\frac{1}{8}$	$\frac{1}{8}$

The statisitical estimators used to summarize the data are:

$\bar{R}$ = sample average of response time (previously defined)

SD(R) = sample standard deviation of response time (previously defined)

(3.10) $\bar{M}_1$ = sample average of number of tasks in, and waiting in queue for, server 1

$$= \frac{1}{T} \int_0^T (\text{number in queue 1} + \text{number in server 1})\, dt$$

where T is the simulated time interval of the experiment

$\bar{M}_2$ = same definition as $\bar{M}_1$ except that queue 2 and server 2 replace queue 1 and server 1

$\bar{B}$ = least squares estimate of slope of conditional response time (previously defined)

Section 2.4 contains exact expressions for the expected values of $\bar{M}_1$ and $\bar{M}_2$ (equation 2.52), and for the mean value of $\bar{B}$ (equation 2.55). The unconditional mean response time, E(R), may be obtained from equation (2.55) by remaining the condition on V. Since this equation is linear in v, one simply replaces v with its expected value E(V). The results of Version I verify this analysis and provide insight into the variability of the results. Version II is included to test the effects of changing the distributions of the overhead delay and the processing requests. The next section contains comparisons of these two versions of TSMOD2 with each other and with the analytic expressions of Section 2.4.

3.4.2 The Results

The results of the simulations of TSMOD2 are presented in a manner similar to that of Section 3.3. Table 3.4 summarizes the results of Version I by presenting the 95 percent confidence intervals for the data and the analytic results from Section 2.4 for these variables. Table 3.5 displays the results for each of the variables for the 20 experiemntal runs of TSMOD2 - Version I. As in the previous section, each run represents 1000 observations obtained after an initialization period of 100 tasks. All of the analytic results lie within the confidence intervals. To test the prediction of a linear relationship between service request and mean response, each task again recorded both its service request and its transit time through the system. Table 3.6 contains the results of separating the service requests into equal intervals and calculating the sample averages for each interval. Figure 3.3 is a graph displaying this data and the 95 percent confidence intervals for response as a function of service request.

Variable	95 percent confidence interval {see eq. (3.4)}	Sample Average	Analytic Result
$\bar{M}_1$	$3.58 \leq E(M_1) \leq 4.29$	3.94	4.00
$\bar{M}_2$	$3.59 \leq E(M_2) \leq 4.45$	4.02	4.00
$\bar{R}$	$7.24 \leq E(R) \leq 8.61$	7.93	8.00
B	$7.89 \leq E(B) \leq 9.20$	8.55	8.88

Table 3.4

Comparison of Experimental and Analytic Values for TSMOD2 - Version I

Experiment	$\bar{R}$	SD(R)	$\bar{B}$	$\bar{M}_1$	$\bar{M}_2$
1	7.26	7.91	7.56	3.62	3.92
2	8.89	9.94	9.22	4.42	4.53
3	7.88	9.30	8.26	3.97	3.78
4	6.13	7.06	6.90	3.17	2.76
5	8.04	10.40	9.54	4.23	3.98
6	7.26	7.90	7.71	3.93	3.58
7	6.89	7.52	7.59	3.17	3.49
8	6.91	7.38	7.06	3.24	3.71
9	8.50	9.54	9.23	3.92	4.62
10	7.11	7.66	7.33	3.63	3.65
11	5.25	5.95	6.10	2.41	2.48
12	11.18	13.15	11.02	5.04	6.44
13	9.43	11.26	10.52	4.85	4.89
14	8.19	8.97	8.44	4.39	3.86
15	8.47	9.25	9.03	4.14	4.20
16	8.41	9.75	9.21	3.94	4.19
17	8.90	10.39	9.32	4.29	4.67
18	8.01	9.59	8.95	4.26	3.74
19	5.59	6.41	6.49	2.65	2.69
20	10.29	11.46	10.93	5.45	5.26
$\bar{X}$	7.93	9.04	8.55	3.94	4.02
s^2	2.15	3.33	1.97	.58	.83
s	1.46	1.82	1.40	.76	.91
$(s^2/20)^{\frac{1}{2}}$	.33	.41	.31	.17	.20

Table 3.5

Simulation Results for Experiments on TSMOD2 - Version I
(Input parameters are displayed in Section 3.4.1)

Experiment	.1 ≤ v ≤ .2		.3 ≤ v < .4		.5 ≤ v < .6		.7 ≤ v < .8		.9 ≤ v < 1.0	
1	2.07	51	4.44	46	5.51	27	6.57	26	10.11	19
2	2.50	58	3.75	25	6.81	23	7.30	37	8.84	16
3	1.91	46	3.46	42	5.37	31	7.09	28	8.38	23
4	1.69	56	3.38	52	4.78	28	4.73	28	7.90	20
5	2.44	45	4.69	36	5.94	26	8.01	18	9.72	14
6	1.73	124	3.30	88	5.02	66	5.94	52	9.03	41
7	1.61	104	2.72	90	4.99	74	4.90	53	7.08	42
8	1.64	89	2.96	80	4.22	66	6.47	57	7.41	38
9	1.97	104	3.71	87	4.95	67	7.00	59	9.47	38
10	1.54	129	4.00	71	4.70	71	6.32	55	7.80	47
11	1.22	141	2.54	72	3.94	70	4.52	56	5.82	51
12	2.95	102	4.90	74	7.50	74	10.67	61	11.63	42
13	1.93	116	4.28	74	5.71	67	8.07	55	10.00	36
14	1.85	110	3.00	84	5.83	69	6.77	53	9.07	39
15	1.91	121	3.18	93	5.46	73	6.63	53	9.55	34
16	1.76	112	3.42	85	4.50	61	7.46	53	10.49	50
17	1.75	106	3.97	75	4.74	66	9.50	59	9.43	42
18	1.81	121	4.02	74	5.25	47	6.30	59	8.56	47
19	1.34	118	2.45	73	3.80	66	5.36	62	5.78	39
20	2.33	123	4.29	87	6.94	67	8.91	50	11.79	44
$\bar{X}$	1.90		3.62		5.30		6.95		8.89	
s^2	.17		.49		.94		2.50		2.63	
s	.41		.70		.97		1.58		1.62	
$(s^2/20)^{\frac{1}{2}}$	.09		.16		.22		.35		.36	

Table 3.6

Average Response Time as a Function of Service Request - TSMOD2 - Version I

Note: Within each interval of service request (v, measured in seconds), the first column contains the experimental average of response times for jobs having requests in the interval, and the second column contains the number of such jobs for each experiment.

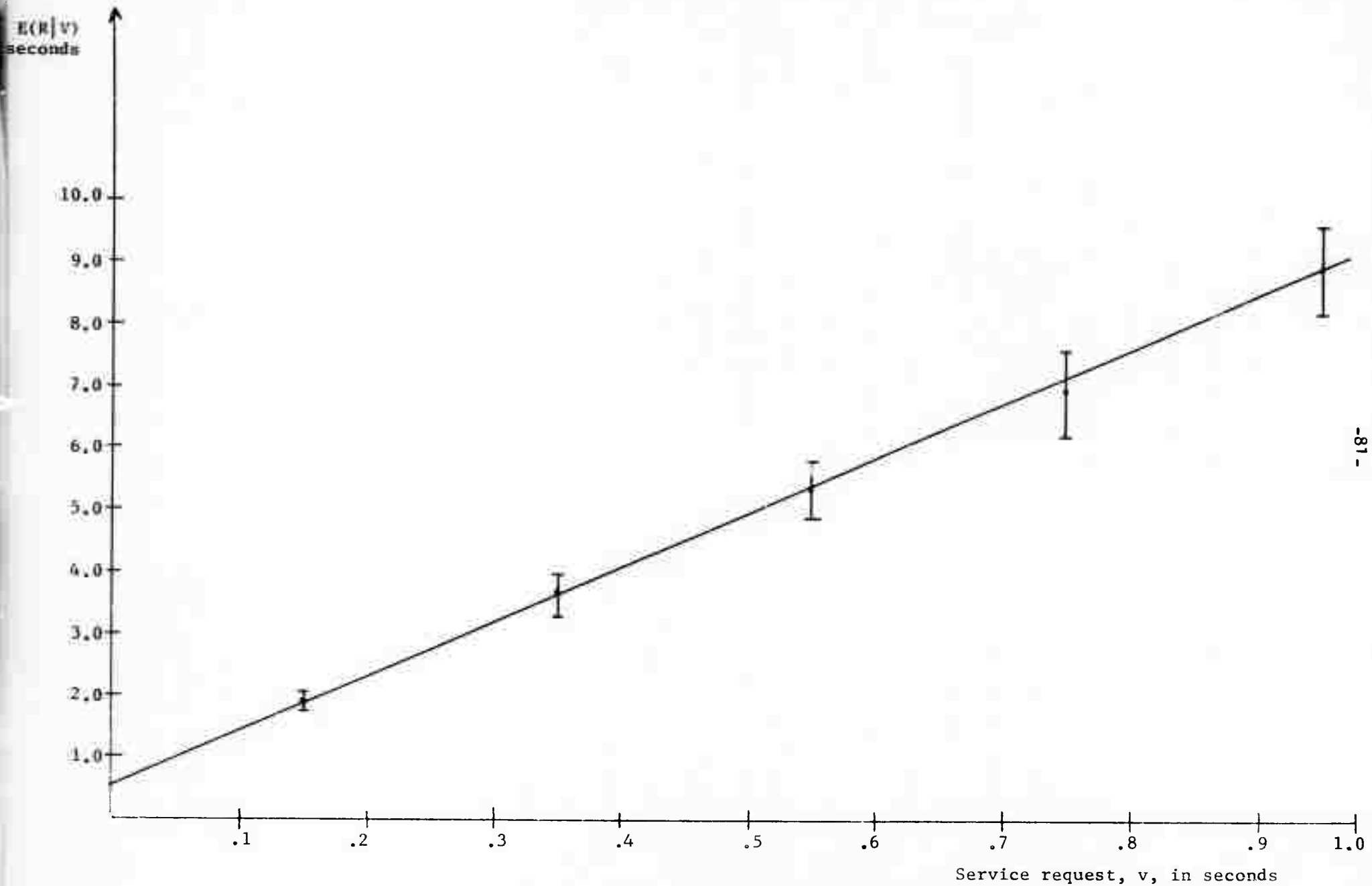


Figure 3.3

Expected Response Conditioned on Service Request - TSMOD2 - Version I

Table 3.7 contains the experimental results for TSMOD2 - Version II. Each run represents 1000 observations obtained after an initialization period of 100 tasks. To test the prediction of a linear relationship between response and service request each task again recorded both its service request and its transit time through the system. Table 3.8 contains the sample averages of response time for various service request intervals and the number of data points in each interval. Figure 3.4 is a graph of this data including the 95 percent confidence intervals for response time as a function of service request.

TSMOD2 - Version II differs in a number of ways from Version I. Since neither the central processor nor the I/O system have exponential service distributions, the outputs and thus the resultant inputs to both systems are not Poisson. Both service distributions have a coefficient of variation less than an exponential and thus the outputs from the servers are more regular than from a Poisson process. This increased regularity of service and input processes reduces both the congestion in the system and response times. The analysis based on exponential input assumptions overstates congestion. Queueing theory offers little help in the analysis of non-Poisson queues in tandem. For example, if the methods of Section 2.4, in which each subsystem is treated as an independent queueing system, are applied to this example and both subsystems are treated as independent M/G/1 queues (with Poisson input rate $\lambda' = 8$) the mean number in each subsystem would be $E(M_1) = 3.809$ and $E(M_2) = 2.933$. The simulation results show these values to be $\bar{M}_1 = 2.60$ and $\bar{M}_2 = 2.38$, or about 25 or 30 percent less than predicted by an M/G/1 model. This example indicates how non-radical changes in modeling assumptions may force the analyst to switch from analytic techniques to simulations to get more accurate estimates of system parameters. Note, however, that the linear relationship between

Experiment	$\bar{R}$	SD(R)	$\bar{B}$	$\bar{M}_1$	$\bar{M}_2$
1	5.93	7.11	13.74	3.09	3.03
2	4.66	5.54	11.61	2.38	2.21
3	4.05	4.54	9.82	2.02	1.83
4	4.84	5.41	11.33	2.49	2.61
5	5.05	5.66	13.11	2.85	2.53
6	4.41	5.39	11.79	2.22	2.15
7	4.42	5.09	11.02	2.25	2.07
8	5.10	5.71	12.25	2.75	2.60
9	4.42	4.88	11.07	2.43	2.00
10	3.79	4.24	9.21	1.98	1.66
11	6.23	7.35	14.55	3.50	3.08
12	4.68	5.44	12.09	2.42	2.34
13	5.36	5.7	11.80	2.87	2.40
14	4.93	5.8	12.60	2.40	2.41
15	6.06	7.7	14.91	3.31	2.58
16	4.42	5.6	11.61	2.28	2.05
17	5.97	7.15	14.49	3.37	2.86
18	5.45	6.16	12.71	2.88	2.59
19	4.00	5.12	10.03	1.94	1.95
20	5.32	5.78	12.26	2.56	2.62
$\bar{X}$	4.95	5.77	12.10	2.60	2.38
s^2	.52	.85	2.38	.21	.15
s	.72	.92	1.54	.46	.39
$(s^2/20)^{\frac{1}{2}}$	.16	.21	.35	.10	.09

Table 3.7

Simulation Results for Experiments on TSMOD2 - Version II
(Input parameters are displayed in Section 3.4.1)

Experiment	.1 ≤ v < .2		.3 ≤ v < .4		.5 ≤ v < .6		.7 ≤ v ≤ .8		.9 ≤ v < 1.0	
1	2.25	92	6.26	57	7.72	37	13.18	18	14.82	11
2	1.57	87	4.24	65	6.24	29	8.29	28	14.20	12
3	1.57	112	3.36	48	6.90	42	7.31	15	9.50	14
4	1.82	101	4.51	47	6.15	31	9.04	19	12.26	5
5	2.00	97	4.24	51	6.31	36	10.89	18	10.13	11
6	1.67	103	3.94	59	5.51	34	7.43	21	9.11	14
7	1.86	91	4.15	57	5.72	25	9.19	28	8.50	12
8	2.06	97	4.72	63	6.20	34	9.77	23	11.77	12
9	1.69	87	4.27	61	6.27	42	9.87	19	11.70	9
10	1.41	106	4.20	50	5.99	34	8.29	27	12.44	14
11	2.17	91	5.14	48	9.91	34	12.71	15	14.29	10
12	1.83	101	3.88	54	6.31	34	9.09	14	11.34	18
13	2.12	84	4.59	62	7.61	34	8.08	18	10.75	11
14	1.92	109	3.72	64	6.23	27	7.53	19	11.02	13
15	2.00	103	5.45	55	7.58	29	11.47	21	16.29	7
16	1.60	91	4.00	58	7.00	29	10.93	15	9.04	12
17	2.19	97	5.54	52	9.09	48	10.80	13	13.83	11
18	2.06	93	4.33	61	7.55	43	10.88	15	9.93	13
19	1.74	115	3.82	60	6.83	30	8.92	13	11.22	10
20	1.89	96	4.21	62	5.99	32	9.44	18	17.12	1
$\bar{X}$	1.87		4.43		6.86		9.66		11.96	
s^2	.056		.49		1.26		2.83		5.97	
s	.24		.70		1.12		1.68		2.44	
$(s^2/20)^{\frac{1}{2}}$	.053		.16		.25		.38		.55	

Table 3.8

Average Response Time as a Function of Service Request - TSMOD2 - Version II

Note: Within each interval of service request (v, measured in seconds), the first column contains the experimental average of response times for jobs having requests in the interval, and the second column contains the number of such jobs for each experiment.

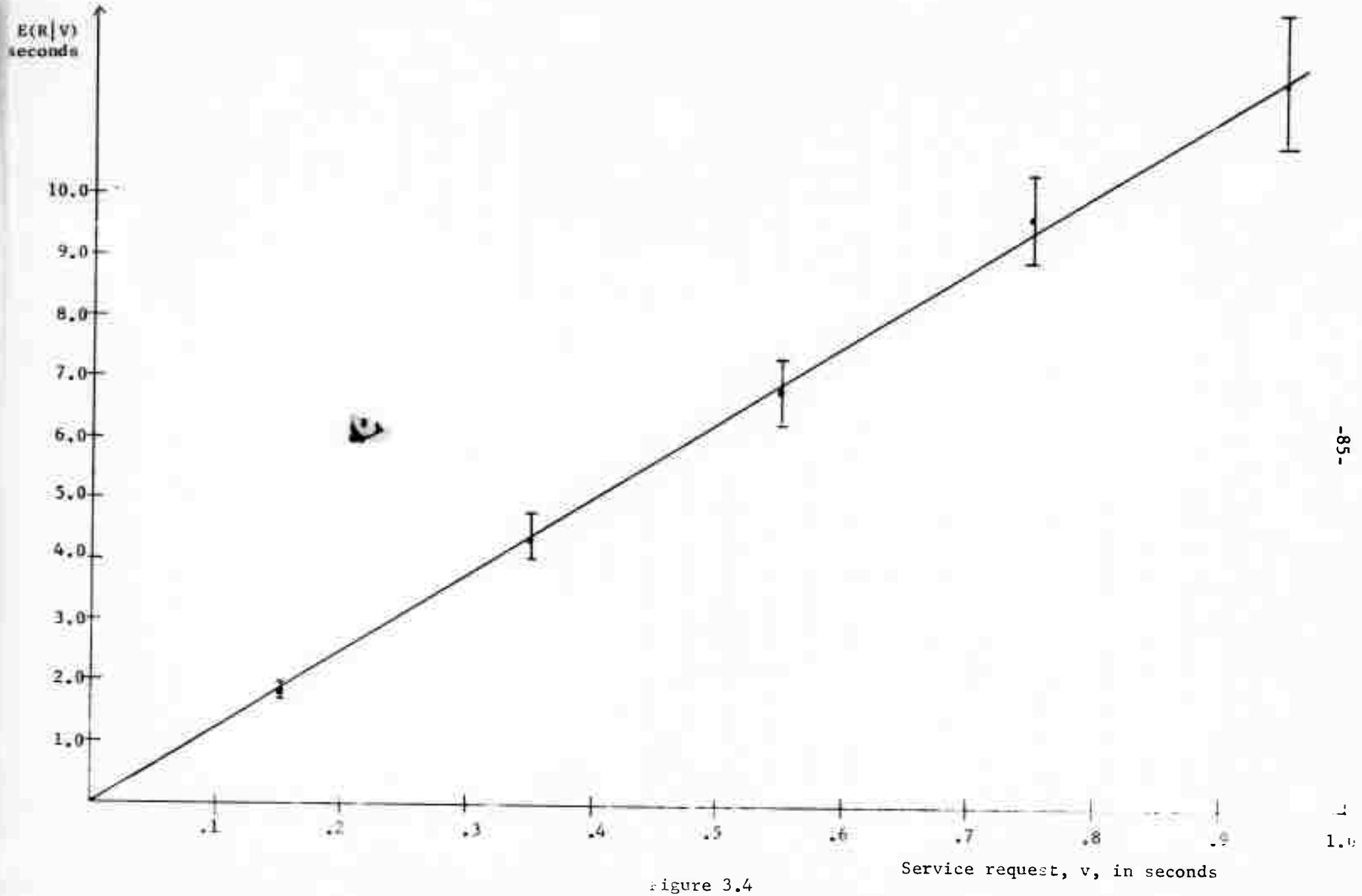


Figure 3.4

Expected Response Conditioned on Service Request - TSMOD2 - Version 1

response and service request, which was derived on the basis of the assumptions of Version I, holds for Version II. Both models exhibit the same general behavior as the input rates increase, or as one subsystem changes its processing capacity relative to the other.

3.5 A SIMULATION OF SCHEDULING IN TSS/360

3.5.1 The Model

TSS/360 is a time-sharing operating system for the IBM 360 Model 67. (IBM, 1968). This computer differs from the standard 360 line because of hardware additions designed to create a virtual memory addressing structure utilizing segments and pages . Section 5.3 contains an elementary discussion of paging. The interested reader is directed to Wilkes (1968) and Denning (1970) for excellent treatments of the subject.

The algorithm which schedules and dispatches tasks in this multiprogrammed, time-shared environment is a section of TSS called the table driven scheduler. Since each TSS system has a different user community to satisfy, and a different hardware configuration, the parameters in the table driven scheduler may be set by each installation. Many parameters within the table are branching codes to other sections of the table. The scheduling section of this simulation is much more detailed than other subsystems such as the paging disks. This model is also implemented in SIMULA. Appendix C contains a listing of the program.[3]

McCredie and Schlesinger (1970) describe the structure of the model in more detail than is required here. One goal of the design was to show that a useful model can be easily implemented without detailed modeling of all system components. Different system modules have vastly different

[3] This program was designed jointly by J. McCredie and S. Schlesinger. S. Schlesinger implemented and debugged the version presented in Appendix C.

levels of detail in the simulation, and the areas of detail may change as the model evolves. Elements of the physical system were included only when necessary because of interactions with the scheduler. The primary goal of the study was to obtain measures of response times experienced by interactive users. Figure 3.5 illustrates the model's structure.

Three hardware facilities appear: the CPU, the paging devices, and memory. The CPU appears implicitly in all software elements of the system and in the execution of user programs. No CPU characteristics such as clock cycle time or instruction times are included, although they are implicit in the amount of computation time used by user programs.

Two types of paging devices are included in the model: disks and drums. Disks are viewed as an infinite source of new pages demanded by executing programs and as an infinite storage facility for pages written out by the monitor. Actual operation of these units is complex since arm seeks on different spindles can be overlapped, and software disk management routines try to optimize arm movements to maximize the flow of pages into core. Instead of modeling this process directly, the access time of a page is drawn from a distribution. The statistical characteristics of this distribution reflect the operation of the actual system. The parameters of the distribution were determined by observations from system logging information. Drums are represented by their revolution times and their capacity in pages. The distribution of access times for pages from a drum is uniform from zero to the revolution time.

There are two major software routines in the model - the timer interrupt handler and the table driven scheduler. Minor software functions occur implicitly in other parts of the model. A timer interrupt occurs

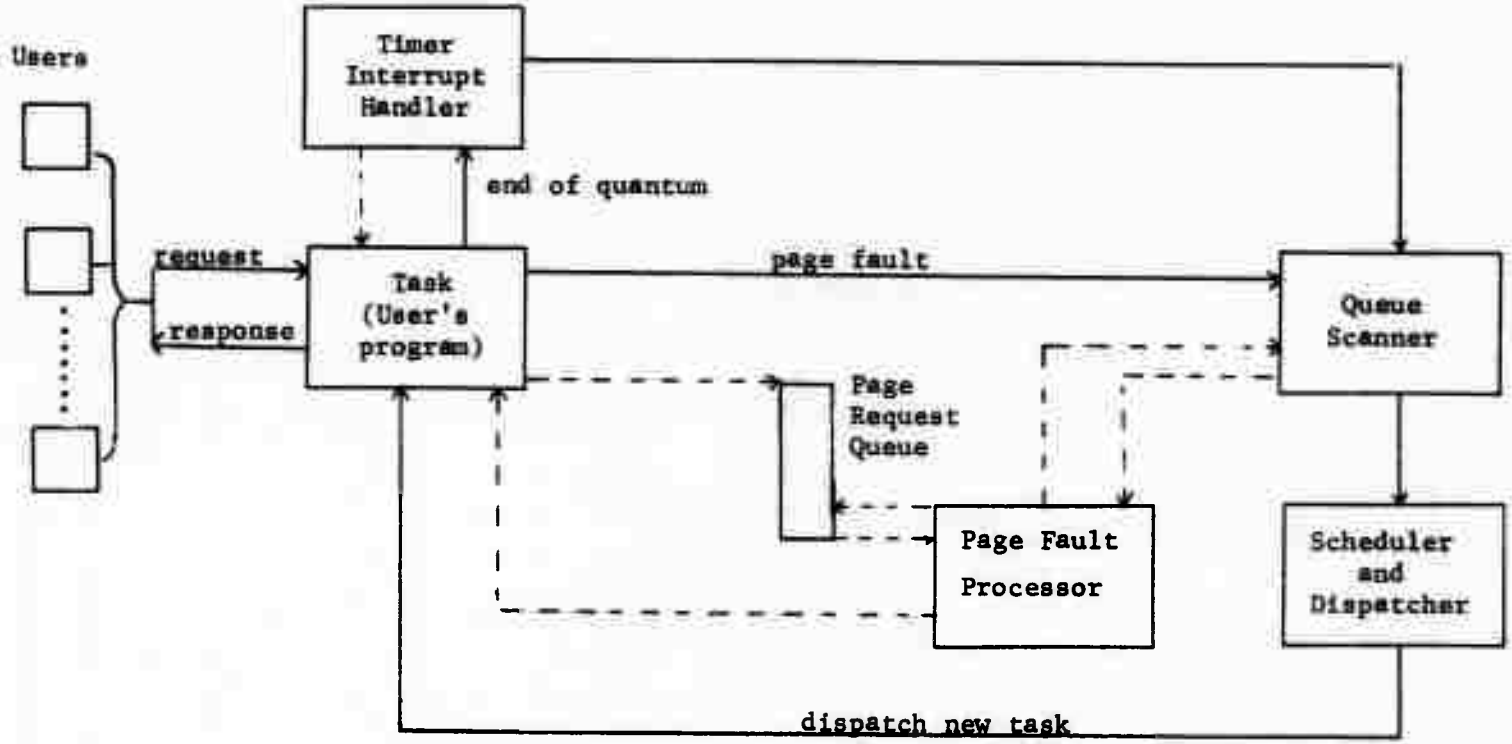


FIGURE 3.5

TSS/360 Model Structure

when a user program has CPU control at the end of its time quantum. The interrupt handler may, depending on the program's scheduling parameters, do any of the following: force a time-slice end and write pages onto the disk (or drum); change the scheduling parameters; give the program an additional time quantum. The timer routine in the model performs the functions of several subroutines of TSS/360 which are called when a timer interrupt occurs.

The table driven scheduler is the most detailed portion of the simulation. Each program in the system is assigned an entry in the schedule table. This entry contains maximum limits on CPU usage and paging activity of a program. A program exceeding the limits is penalized by loss of eligibility for CPU allocation and possible lowering of priority. This penalty occurs by changing the program's schedule table entry to a new one depending upon how the program exceeded the bounds of its previous entry. For each maximum there is a new schedule table entry to which the program will be assigned if that limit is exceeded. The monitor interrupts a program during its time slice to check if any bounds have been violated.

The scheduler maintains several lists of programs, each one having a different eligibility for CPU allocation. User programs move among the lists depending on their schedule table entry and operating characteristics. The schedule table in the actual system has limits on additional operating characteristics of programs to enable fine-tuning of the scheduling algorithm.

Program behavior is characterized by periods of CPU usage separated by page faults. When the program receives CPU control from the scheduler it is interrupted only by a timer interrupt or page fault. While it has

CPU control, no classification is made of language used, system functions called, or other modes of activity. The only parameter of interest during CPU usage is the time necessary to complete the user request. Paging activity is based upon the working set concept of Denning (1968). Each request specifies the working set size for that request. The user program then calls for sufficient pages to fill the working set. There is no distinction concerning the contents of each page. Only the number in core is of interest.

The users in the model make terminal requests and wait for system response at the terminal. A request consists of the amount of CPU time required and the number of new pages which have to be brought into core for a complete working set. The model draws both of these parameters from distributions either approximating observed system behavior or testing hypothetical modes of system use. The distribution of response time experienced by a user is the statistic of primary interest.

3.5.2 The Results

A modular simulation such as the one described in the preceding section gives a systems analyst a great deal of flexibility. The original goal of the model was to aid in tuning TSS/360 to the job load of the CMU environment. The first use of the simulation involved a group of users who proposed paying a higher rate to receive better service from the system. The plan designed to achieve this service differential was to create an algorithm in the schedule table so that users from the higher priority class would follow a different path through the schedule table. Tests using the model indicated that benefits of the proposed solution were marginal compared to the increased charge. A different plan, based upon the concept of guaranteed terminal access to a certain

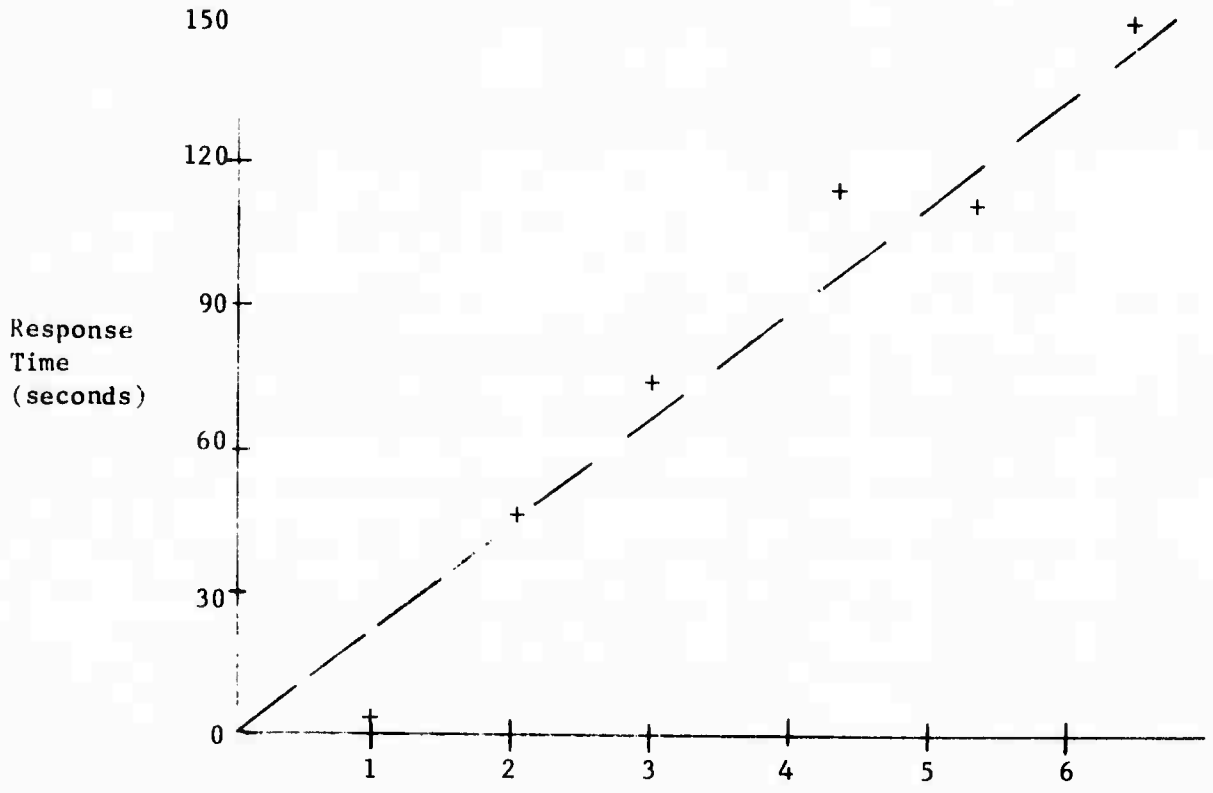


Figure 3.6

Model of TSS/360 - Heavy Usage Script

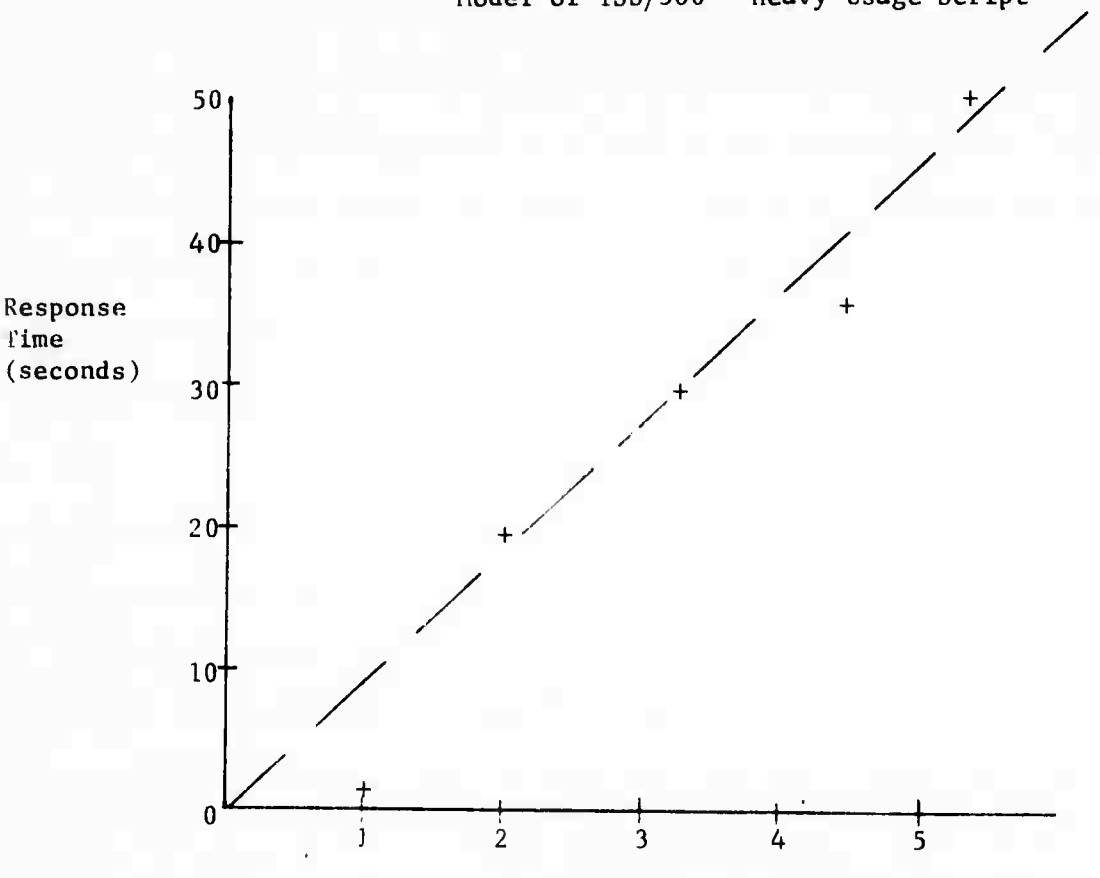


Figure 3.7

Model of TSS/360 - Probe Job in Normal Mix

number of higher priority users, provided the required service differential.

The model may be used like the previous ones in this chapter to measure both the mean value of response times for various loadings and the mean response time conditioned on service request. Figures 3.6 and 3.7 are typical graphs of such experimental results. The first is a heavily loaded system, and the second is more representative of a typical user load. Results from this model of TSS/360 have the same global characteristics as the results from the analytic models of Chapter 2. Mean response conditioned on service request is a linear function of the service request, and severe response degradation occurs with heavy loading.

3.6 DISCUSSION

The examples of this chapter illustrate some of the advantages and disadvantages of simulation as a tool for the analysis of computer systems Flexibility is demonstrated by the ease with which these models may be modified incrementally to examine the effects of changes such as new distribution assemptions or the addition of another processing subsystem for input and output activity. Appendix B contains the listing for TSMOD2. To use this program to study TSMOD1, one removes the activity describing the I/O system, and changes a few lines of code in the activity describing the central processor. Appendix E contains a listing of the program used to study different dynamic control algorithms for time-shared systems. The program used to study TSMOD2 is also the basis for this latter model. The dynamic control algorithm, and the user behavior model, were easily added to the original design. The model of TSS/360 which is listed in Appendix C

also illustrates the flexibility of a modular approach to simulation. Once implemented and debugged, a well designed simulation can be a very powerful aid to system analysis.

Large simulations are expensive to implement and to run. The data in the previous sections illustrate the variability which is inherent in queueing processes. To properly analyze the results of experiments on models of such systems, one must make many experimental runs. Since each experiment is only one realization of a stochastic process, it can become expensive to produce tightly bounded estimates of the values of state variables.

The primary goal of this chapter was to examine the robustness of the analytic models developed in Chapter 2. When the simulations are closely related to the analytic models, the analytic results are within the confidence intervals determined by the experiments. In addition, the global behavior of more complex systems such as TSMOD2-Version 2 and TSS/360 has the characteristics predicted by the analytic models. The next chapter presents the results of three empirical investigations of operational time-shared computing systems.

CHAPTER 4

EMPIRICAL STUDIES OF SYSTEM BEHAVIOR

4.1 INTRODUCTION

Although the models of Chapter 2 include important features often neglected in other models, they are simple structures when compared with actual computer systems. The simulation studies of the previous chapter illustrate a number of properties of feedback queueing structures, and demonstrate that more complex models exhibit the same basic behavior patterns as the simpler analytic structures. The purpose of this chapter is to present empirical evidence that one may describe macroscopic performance measures of actual systems with these same analytic formulations. The models capture enough of the essential features of actual designs to enable one to use them, as outlined in Section 1.1 and illustrated in Chapter 5, in systems analysis studies.

There are many different ways to measure and evaluate the performance of computing systems. Perhaps the oldest, and most common, technique is the execution of a representative set of programs (called a benchmark series) while monitoring the system. The development of a benchmark series which accurately models the characteristics of a particular user community is a difficult problem that is not adequately solved. An additional problem is that a computing system can usually be tuned to do well on any specific benchmark series. However, the purpose of the experiments which follow is not to compare and evaluate different systems for a particular environment, but rather to explore their general behavior and compare it with that predicted by the models of Chapter 2.

A representative benchmark series is particularly hard to create for a time-shared system that is to support many independent users. A method which is related to the standard benchmark technique is a user population simulator which creates pseudo user tasks which follow a script with precise inter-command timing delays. This procedure is an important evaluation technique because of its high degree of repeatability. Two common ways of implementing it are to dedicate a second computer to the task of simulating the user population or to create, within the system being measured, a special program that interfaces with the operating system and is capable of inserting user jobs into the task queue. Special programming systems are required for this technique.

An alternative design, which was used for the first experiment, makes use of people who follow a benchmark script. The purpose of this design is to create a semi-controlled environment in which users can run pre-designed programs and experience realistic system response. The experiment was non-repeatable at the instruction level since users were not synchronized or driven by any timing information, but this type of precision was not required for the global performance measures of interest in this chapter. An advantage of this semi-controlled design is that actual user behavior does not have to be modeled. Real users type at different speeds with different accuracy. To use a user population simulator one must design a script, and an explicit model of the behavior of all users. Thus in the semi-controlled environment one trades a gain in realism for a loss in repeatability. Another reason for using the semi-controlled environment was that the response time investigations, reported in the next sections, were but one dimension of a larger investigation of

properties of different time-sharing systems. In the larger study, qualitative judgments of users along dimensions such as ease of use, perceived power, and reliability were important.

A disadvantage of both of the experimental techniques discussed so far is that to observe the system in a realistic mode of operation one must create a script which is a good approximation to user behavior. An alternative procedure is to measure the system during an actual user session. One may treat the computer and its user community as one large system to be measured by inserting a probe task with known characteristics into the job queue and monitoring its behavior. An advantage of this technique is that if the probe job does not use a great many resources, measurements can be made during actual user sessions without causing severe degradation to other users.

The following experiments include the monitoring of both an operational IBM 360/67 using TSS/360 and a Univac 1108 using EXEC-8. In the first experiment, a set of users followed a script which directed their interactions with TSS/360 while the system calculated and saved response statistics. For the second, a small probe job was inserted into the actual user environment. This probe entered the 360/67 at random intervals over a period of a number of weeks, measured selected state variables, and gathered statistics about its own response. The last experiment was similar to the first, but it was performed on the 1108 using EXEC-8. Thus the same basic experiment was conducted on two very different systems, and two different types of experiments were performed on one of the systems. The following sections contain a more detailed discussion of each of the experiments and their results.

4.2 EXPERIMENTS ON TSS/360

4.2.1 Controlled User Script

The computer used was the Carnegie-Mellon IBM 360/67 using TSS/360 Version 5.1. The experiment was run during the spring of 1970 when Carnegie was using a memory hierarchy which included IBM large capacity storage (LCS) which had a relatively slow cycle time of eight microseconds.

For this experiment 29 selected TSS users received a short training session in which they were introduced to both the goals of the study and the script. The computer was dedicated to the experiment for approximately two hours. Users not participating in the study were denied access to the system. Participants had special account numbers valid only for the study. All background batch computing was terminated so that only the script tasks submitted from terminals would be active.

Figure 4.1 is a block diagram of the script, and Appendix D contains a complete listing of the script. After establishing a connection with the computer each user performed initialization tasks which gave him access to required programs and which created necessary statistical files. The initialization program called the FORTRAN compiler and directed it to process TEST1. There were two small intentional errors in this source code and each person then used the editor to correct these statements and list the program. After the program was recompiled each user called it a number of times submitting as control input the number of times the program should execute its major loop. The code multiplied two thirty by thirty matrices, and stored the result in a third matrix. Each user task measured and printed its response time, and stored the information

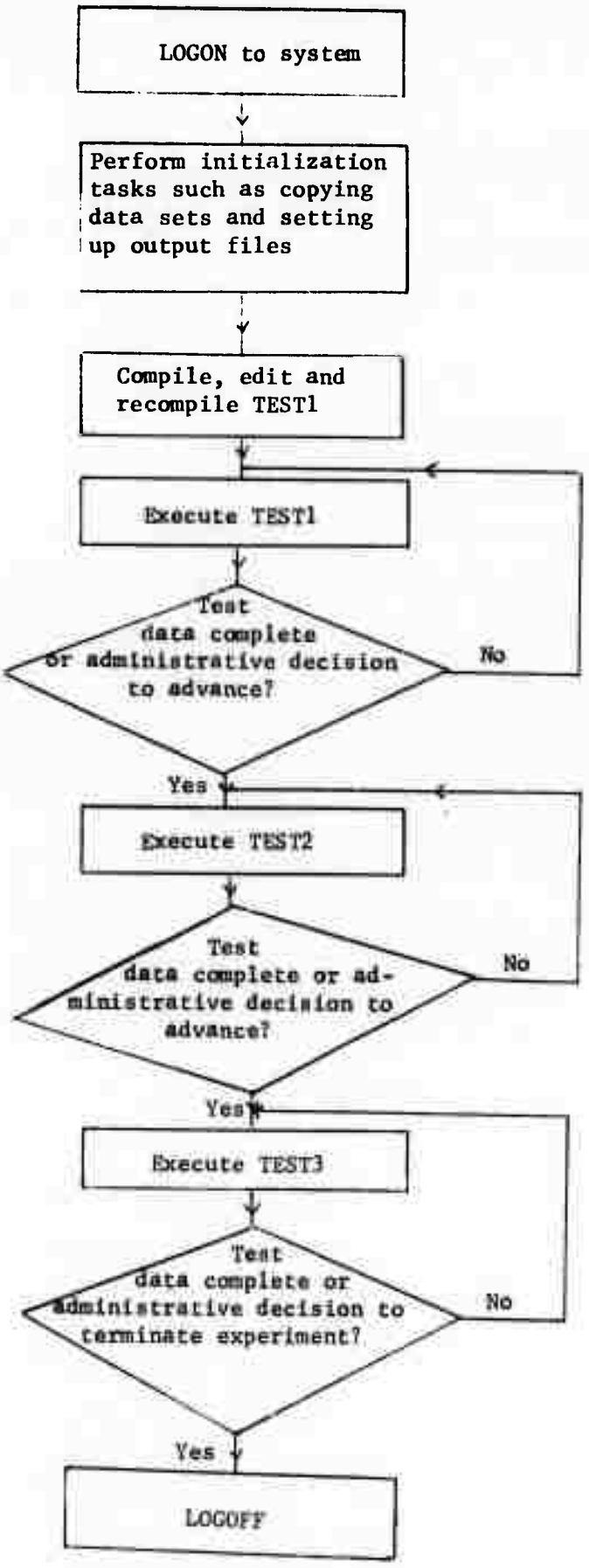


Figure 4.1

Block Diagram of User Script

on a file for later statistical analysis. The second program, TEST 2, was a small job which adds the integer one to a counter a specified number of times and then stops. The third program, TEST3, was similar to TEST1 in computing requirements, but it demanded a large amount of storage (approximately 23,000 words). All of the programs computed the elapsed time required by the system to process each request. The users saved all terminal listings for later data verification purposes.

Figure 4.2 is a graph of the sample averages of response times for the three tests. The horizontal axis measures the computing request in terms of the number of iterations, I, through the major loop of TEST1. Each of the other programs was scaled to this measure of central processor demand. Each point on the graph represents the sample average of requests having the same value of I.

4.2.2 TSS/360 Probe Experiment

The goal of this experiment is to investigate mean response as a function of system load and service request in a typical user environment. Over a period of a number of weeks a version of TEST1, which is listed in Figure 4.3, was submitted to TSS/360 as a conversational task. A supporting package was written to simplify the execution of each run of the probe. One command initiated a complete cycle of five replications of TEST1, each with a different parameter to control the processing request. As in the previous experiment, control information was in terms of the number of required iterations of the major loop of the program. Data points were saved in a master file for later processing. Each data block included the time of the experiment, the number of active conversational

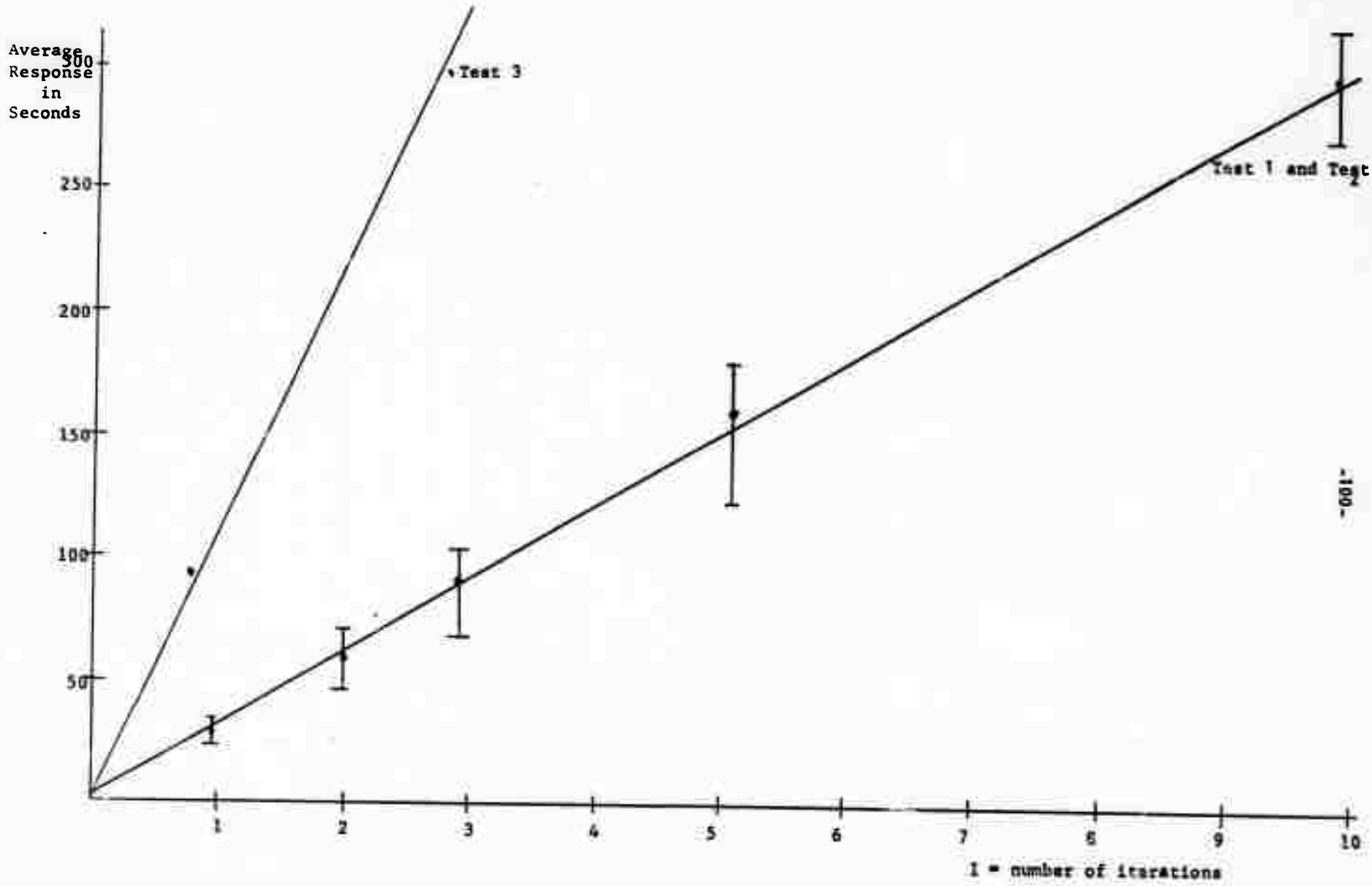


Figure 4.2

Average Response Time as a Function of Service Request - TSS/360

Figure 4.3

Listing of TEST1

```
0000100 DIMENSION IB(8),IE(8)
0000200 DIMENSION M1(30,30),M2(30,30),M3(30,30)
0000300  100 PRINT 899
0000400 899 FORMAT(/'  TEST1: PLEASE ENTER N IN FORMAT I2')
0000500 READ 900,N
0000525 900 FORMAT(I2)
0000550 IF (N.EQ.0)  GO TO 10
0000600 PRINT 901,N
0000700 901 FORMAT('  TEST1: ITERATION=',I3)
0000800 CALL CLOCK(IB)
0000900 DO 6 L=1,N
0000910 DO 2 I=1,30
0000920 DO 1 J=1,30
0000930 M1(I,J)=2
0000940 M2(I,J)=3
0000950 1 CONTINUE
0000960 2 CONTINUE
0000970 DO 5 I=1,30
0000980 DO 4 J=1,30
0000990 M=0
0001000 DO 3 K=1,30
0001010 3 M=M+M2(I,K)*M1(K,J)
0001020 M3(I,J)=M
0001030 4 CONTINUE
0001040 5 CONTINUE
0001050 6 CONTINUE
0001100 CALL CLOCK(IE)
0001200 TRAN=36000*(IE(1)-IB(1))+3600*(IE(2)-IB(2))+600*(IE(3)-IB(3))
0001300 TRAN=TRAN+60*(IE(4)-IB(4))+10*(IE(5)-IB(5))+(IE(6)-IB(6))
0001400 TRAN=TRAN+.1*(IE(7)-IB(7))+.01*(IE(8)-IB(8))
0001700 PRINT 903,IB
0001701 903 FORMAT('  START TIME= ',2I1,':',2I1,':',2I1,'.',2I1)
0001702 PRINT 904,IE
0001703 904 FORMAT('  END   TIME= ',2I1,':',2I1,':',2I1,'.',2I1)
0001704 PRINT 905,TRAN
0001800 905 FORMAT('  RESPONSE TIME= ',F8.2)
0001900 WRITE(1,800) N,IB,IE,TRAN
0001950800 FORMAT(2H 1,I8,8I1,8I1,F8.2)
0002000 GO TO 100
0002050 10 PRINT 906
0002051 906 FORMAT('  TEST1 NOW COMPLETE. YOU ARE IN COMMAND MODE')
0002052 STOP
0002100 END
```

```
 TEST1: PLEASE ENTER N IN FORMAT I2
01
 TEST1: ITERATION=  1
 START TIME= 20:17:05.86
 END   TIME= 20:17:13.86
 RESPONSE TIME=     8.00
```

```
 TEST1: PLEASE ENTER N IN FORMAT I2
01
 TEST1: ITERATION=  1
 START TIME= 20:14:24.35
 END   TIME= 20:15:19.09
 RESPONSE TIME=    54.74
```

users, control information and the response time experienced by the program. The system operator initiated a run of the probe approximately once every hour while TSS was running user jobs.

The probe ran in a different environment each time it was initiated. The program remained the same, but the system load varied from light to heavy. Detailed status information about each user was not gathered. The data were classified according to the number of active users and the processing request. Figure 4.4 is a graph of the sample average of response time conditioned on service request for a number of different system loads. Each line represents a different user population density. Data points were aggregated into four classes according to the number of active users on the system (5-10, 10-15, 15-20, 20-25). Figure 4.5 treats the same data in a different way. Instead of displaying equal load lines, this graph presents equal service request curves plotted against the number of active users. Since the users were grouped into classes, the data points are in the middle of the appropriate intervals. Again the service request is measured in terms of I, the number of times the probe had to execute its major loop.

4.3 EXPERIMENT ON EXEC-8

EXEC-8 is a general purpose time sharing operating system which runs on the Univac 1108 computer. Many of its design goals are the same as TSS/360. One major difference in philosophy is that there is no virtual memory. Instead of dividing programs into pages, as in TSS/360, EXEC-8 moves entire programs back and forth from core memory to external storage.

This experiment was the same as that described in Section 4.2.1. The overall design was identical to Figure 4.1. The computer had 196K words

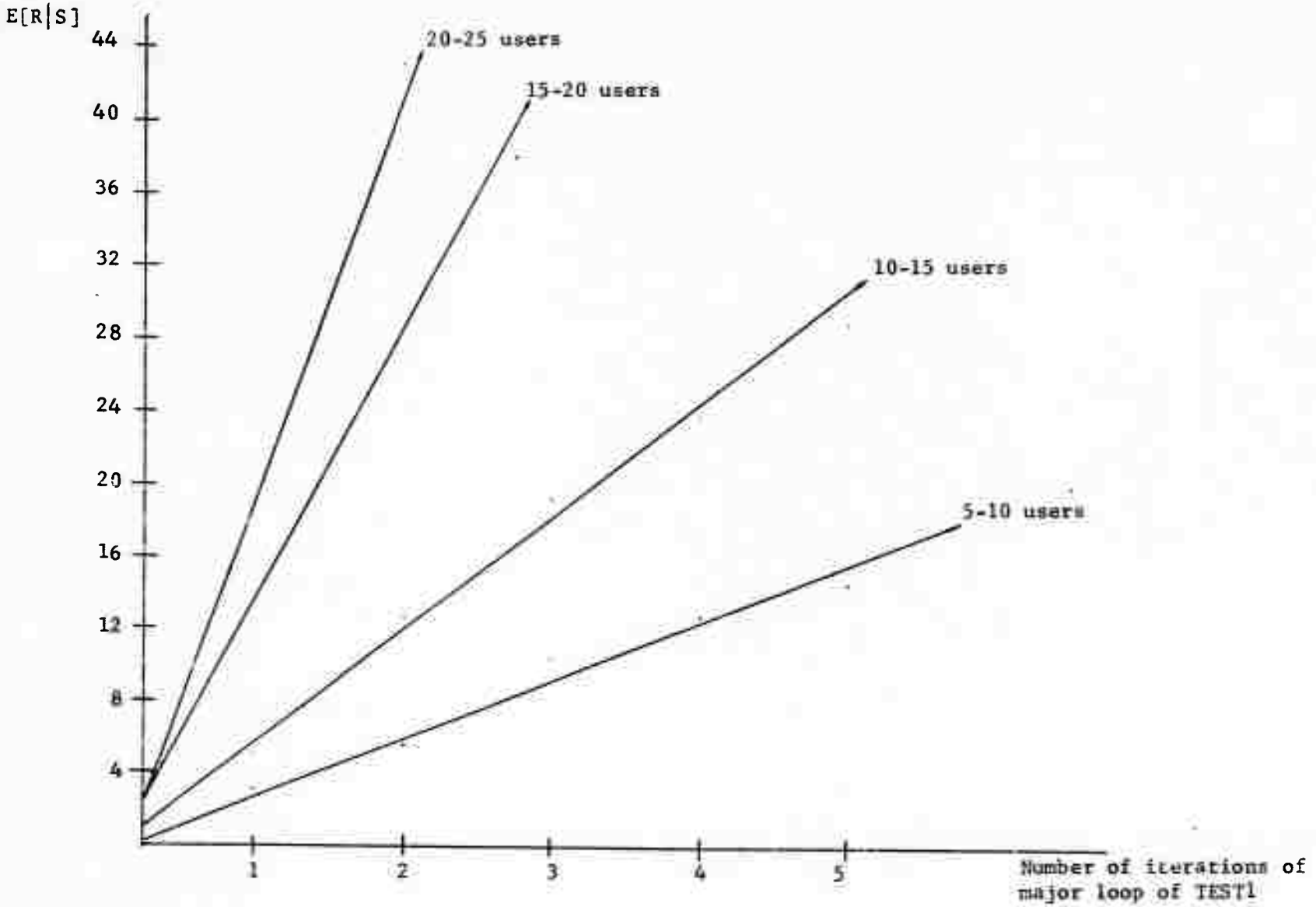


Figure 4.4

Expected Value of Response Time Given the Service Request for TEST1 on 360/67

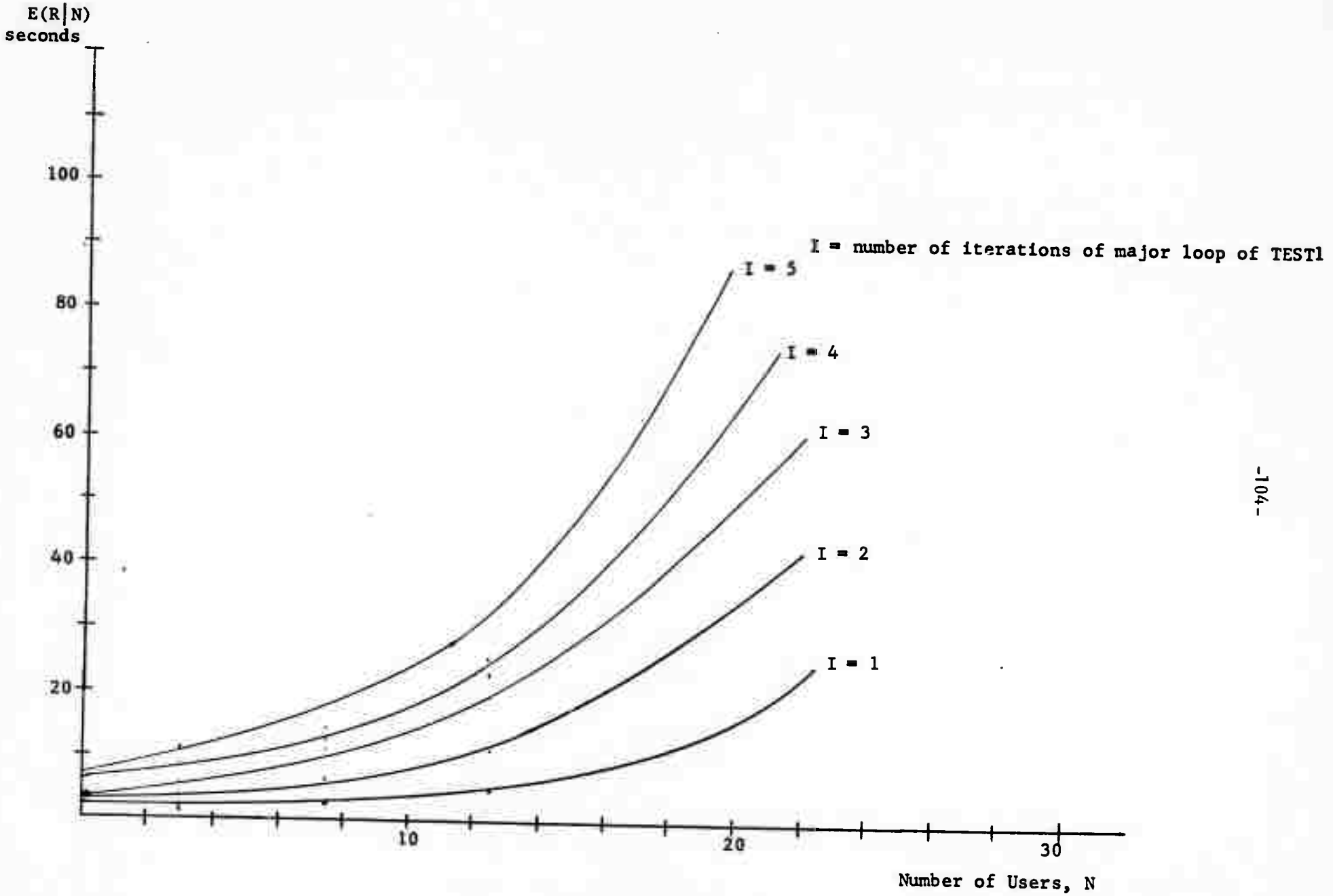


Figure 4.5
Expected Value of Response Time Given the Number of Users on TSS/360

of high speed core storage, and a fully balanced set of drums and disks. The computing system was dedicated to the experiment for approximately two hours. The test was part of an overall evaluation of EXEC-8 performed by the CMU University Computing Council, in the spring of 1970, and Univac donated the time at their Chicago Information Systems Division service bureau. Forth-three users participated in the experiment. Each followed the same script, and each saved the terminal listings for later analysis. Figure 4.6 is the graph of the sample averages of response conditioned on service request. Once again the global behavior corresponds to the predictions of the analytic models.

4.4 DISCUSSION

TSS/360 and EXEC-8 are large operating systems which control complex time-sharing systems. The philosophy of memory organization is completely different in the two designs. The results presented in this chapter should not be used as a basis for comparing the two different systems. There was no attempt made to calculate the costs of the machines, and thus one cannot make cost/performance comparisons. The particular programs used in the script and for the probe could be unintentionally biased towards one machine. The goal of this chapter was not to compare two different systems, but to observe their macroscopic performance.

The models of Chapter 2 capture enough of the basic design philosophy of these two systems to predict the observations that for equal values of load, measured in terms of the number of active users, response is a linear function of service request and that for equal service requests, response is a non-linear function of system load. One important variable not explicitly considered in any of the previous models is the size of

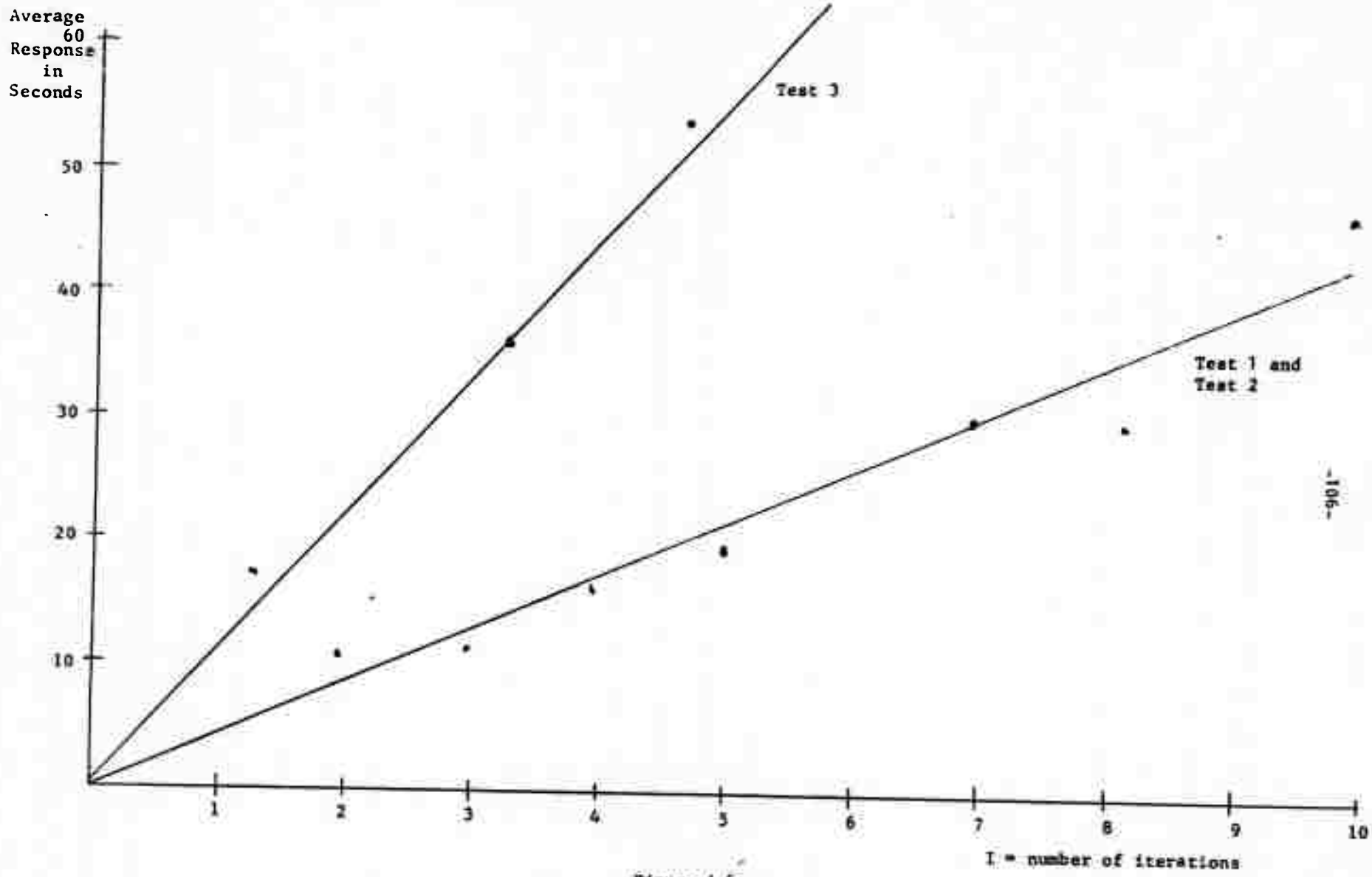


Figure 4.6

Average Response Time as a Function of Service Request - 1108/EXEC-8

user tasks. Figures 4.2 and 4.6 show that for equal service request, measured in instructions from the central processor, response depends on the size of the program. TEST1 required approximately 3,000 words for its code and data, and TEST2 needed only a few hundred words. TEST3, however, was relatively large and required approximately 23,000 words of core storage. New models which consider the memory requirements of user programs should be developed to investigate this observed relationship between response and size.

Analytic models, simulations, and empirical investigations should interact. Section 1.1 outlined a number of ways these tools should aid each other in systems analysis. The results of this chapter clearly indicate that future analytic models should consider memory requirements as an explicit parameter. The results also show that models like those of Chapter 2 do have the ability to predict macroscopic behavior of actual systems. The next chapter illustrates a number of ways these analytic results may be used.

CHAPTER 5

APPLICATION OF THE MODELS

5.1 INTRODUCTION

Chapter 2 contains a number of new results from models of time-shared computing systems. Each model focussed upon a different feature of real systems because the current state of queueing theory makes the simultaneous treatment of all such features very difficult. The goal of this chapter is to illustrate how models of this level of complexity may be used in three areas: (1) design (2) performance improvement studies and (3) dynamic system control. The user of such models need not have the mathematical background to develop new solutions, but he must understand the underlying concepts and assumptions. The developer of new models should present them in such a way as to make these features readily understandable to the user. The presentation of Chapter 2 was an attempt to make the assumptions and the methodology obvious. The experiments of Chapters 3 and 4 are included to demonstrate that systems of greater complexity than the models, including real time-shared computers, behave in ways which the models predict. This chapter presents some realistic examples of how these models may be used.

5.2 SOFTWARE LOCKOUT IN A MULTI-PROCESSOR

The following problem illustrates the use of modifications of the developments of Section 2.4 and Section 2.5 in a real design problem. The Department of Computer Science at Carnegie-Mellon University is implementing both a multi-processor system called C.mmp (Bell et al., 1971) and an operating system for it called HYDRA (Wulf et al, 1971). The major goal of this project is to

create a powerful and flexible computing system which will support parallel and pipeline processing and which is capable of orderly growth through the addition of processors and memory. A major problem in the architecture of such a system is the scheduling and coordination of the many individual processors. The approach taken in HYDRA is to have a common shared data base which contains all of the information necessary for a processor to make a scheduling decision. While one processor is examining or updating this shared information all others must be prohibited from accessing or changing it. The act of protecting data from all but one processor is called locking and the code accessing this data is called a critical section.

Figure 5.1 is a diagram of the model used to study the locking of critical sections in C.mmp. Each of N homogeneous processors is a source of scheduling requests, and each request must gain access to all of the data starting with the first critical section and proceeding to all sections in order. If the first critical section is free, the processor issuing the request locks the section and uses (possibly modifying) the data. When the request is satisfied, the processor unlocks the first critical section and tries to gain access to the second. If a critical section is locked, the processor must wait for access until it has been unlocked by some other processor. Thus, a queue of waiting processors may form in front of each critical section. A processor will be designated as "blocked" for a scheduling operation if it is either waiting for, or inside, one of the S critical sections.

A basic design problem is to determine how many critical sections, S, to build into the shared data base. At one extreme S could equal the number of logically distinct information segments in the data base, and at the other extreme it could be equal to unity by having a single critical section include the entire data base. There is an overhead loss of L time units associated with each locking and unlocking operation which would be minimized by setting

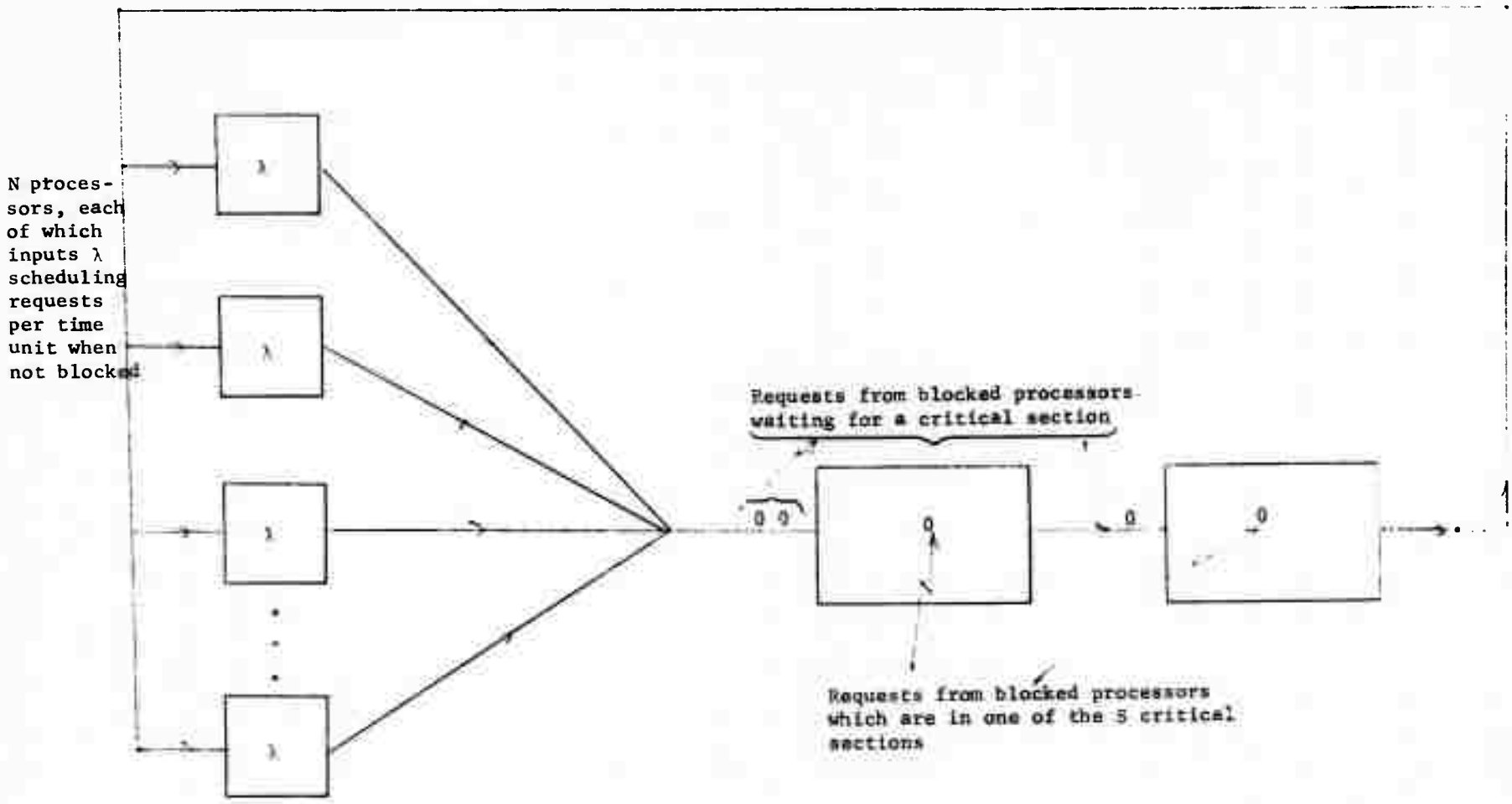


FIGURE 5.1

Critical Section Locking in C.mmp

S equal to one, but this solution does not allow for concurrent use of the scheduling information. Overall system efficiency may be increased significantly by allowing many processors to have simultaneous access to the scheduling information even though the additional locking operations introduce more overhead.

The total time, T, that a processor must have access to the shared information in order to make a scheduling decision is a random variable due to the dynamic nature of the data base and to the many different kinds of decisions. Let the time spent in each of the S critical sections be an exponentially distributed random variable with mean L + T/S. This approximation includes an assumption that the designer would try to balance the system so that processors would spend the same mean time in each critical section.

5.2.1 A Poisson Source Tandem Queueing Model - MOD1

A simple but useful model to explore this problem is the Poisson source, tandem queueing structure discussed in Section 2.4. Since there is no cycling of requests within the critical sections, the appropriate modification of this model is to set ℓ, the probability that a request will leave the system after passing through the S servers, to unity. All arguments presented in Section 2.4 to justify analysis based on independent M/M/1 queueing systems apply to this model also.

Let the source of scheduling requests be Poisson with rate $\lambda \cdot N$ requests per unit time where N is the number of processors making requests, and λ is the rate from an individual processor. A major assumption in this formulation which is not realistic is that a processor will continue to issue scheduling requests before the previous ones have been satisfied. Congestion will be

more severe in this model than in a more realistic formulation which assumes that a processor may not generate additional scheduling requests when it is in the blocked state. The next section considers such a finite source model. The mean number of requests, $E(M_i)$, either waiting for, or inside of, each of the S critical sections is simply the expected number of tasks in an M/M/1 queueing system with input rate $\lambda \cdot N$, and service rate $1/(L + T/S)$.

$$(5.1) \quad E(M_i) = \lambda \cdot N / \{1/(L + T/S) - \lambda \cdot N\}$$

The total number of requests waiting for, or inside of, all critical sections is $S \cdot E(M_i)$. Little's theorem, presented in equation (2.12), relates the mean response time through all critical sections, E(R), to the mean number of blocked requests.

$$(5.2) \quad E(R) = S.E(M_i)/(\lambda \cdot N) = S/\{1/(L + T/S) - \lambda \cdot N\}$$

Figure 5.2 is a graph of mean response through all critical sections for different values of S and N for fixed λ, L, and T for the Poisson source model.

5.2.2 Finite Source Models - MOD2

A more realistic approximation to actual processor behavior than the previous model is the assumption that each of the N processors computes for an exponentially distributed random interval and then makes a scheduling request. The processor will be unable to proceed with normal computing until it has gained access to all S critical sections and is no longer in the blocked state. As in the previous model, the time spent in each critical section will be an exponentially distributed random variable with mean $L + T/S$.

If the number of critical sections equals one, the model reduces to one form of the machine repair problem discussed in Section 2.5 (with the overhead

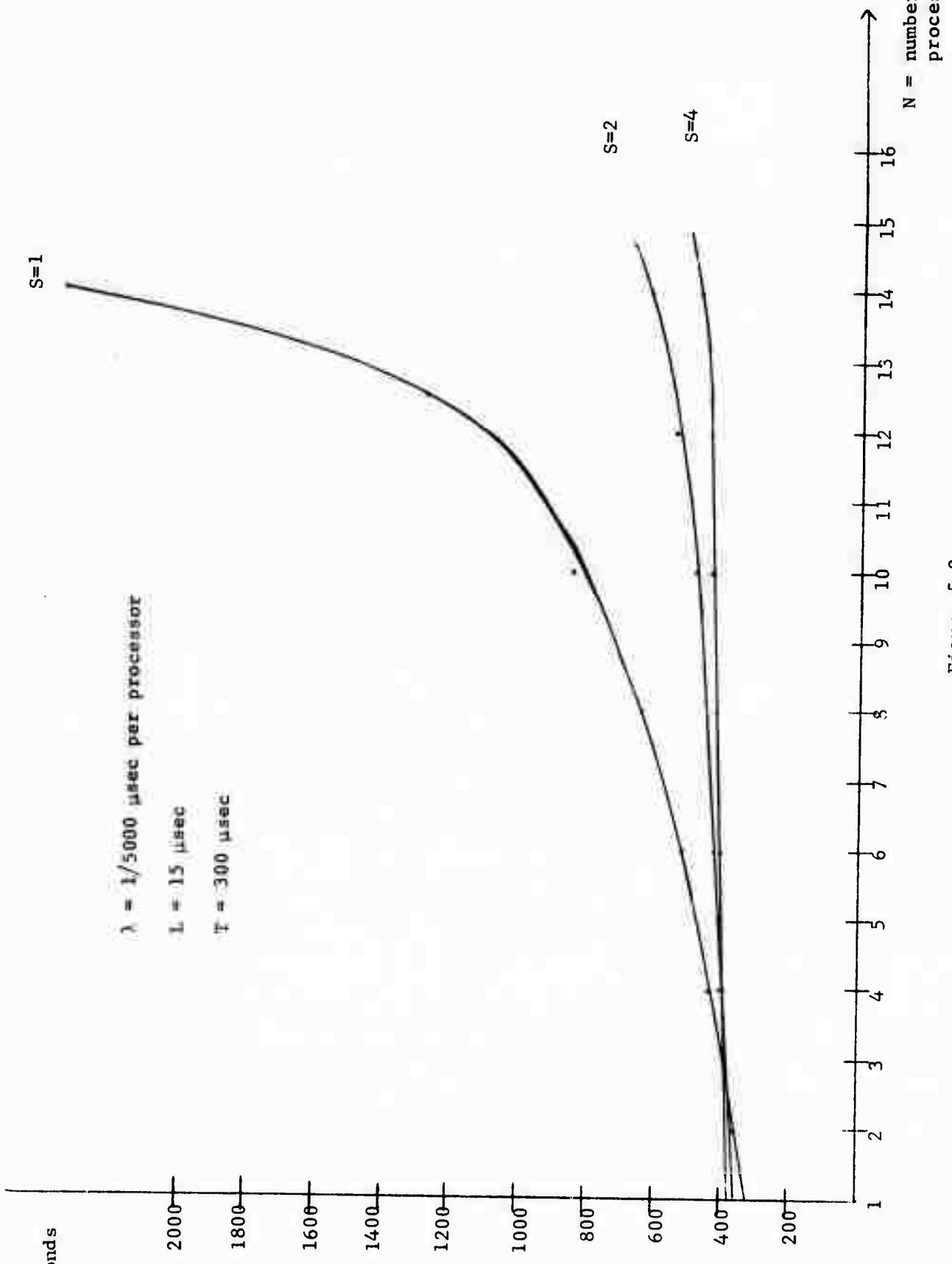


Figure 5.2

Mean Response Time for Scheduling Operations - MOD1

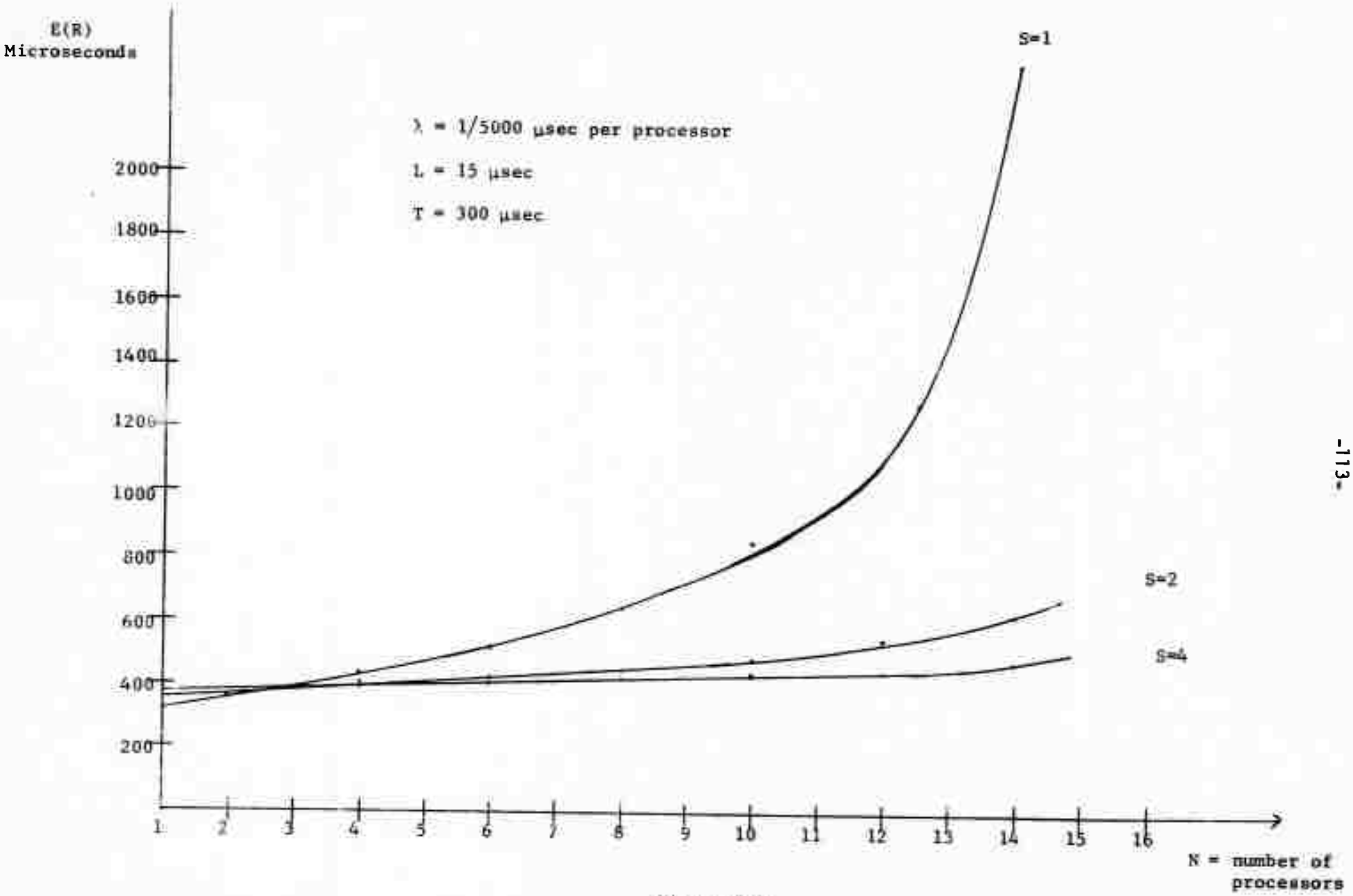


Figure 5.2
Mean Response Time for Scheduling Operations - MOD1

loss fraction, f, equal to zero). Madnick (1968) used this model to calculate the mean number of blocked processors, when S=1, as a function of P, the number of processors in the system. Jackson (1963) derives results for general networks of Poisson queues which have state dependent inputs. This general solution may be applied to this model in a straightforward manner to determine the mean number of blocked processors for an arbitrary number of critical sections, S. His method leads to the following equation for the mean number of blocked processors in the finite source model illustrated in Figure 5.1.

$$(5.3) \qquad E(M) = \frac{\sum_{i=1}^{N} \left\{ \frac{i \cdot N!}{(N-i)!} \cdot \binom{S-1+i}{S-1} \cdot \left(\frac{\lambda}{\mu}\right)^i \right\}}{\sum_{i=0}^{N} \left\{ \frac{N!}{(N-i)!} \cdot \binom{S-1+i}{S-1} \cdot \left(\frac{\lambda}{\mu}\right)^i \right\}}$$

Where $\mu = 1/(L + T/S)$

The second argument used to derive (2.69) applies to this model and thus equation (2.70) may be used to relate mean response time through the critical sections to mean number of blocked processors.

$$(2.70) \qquad E(R) = \frac{E(M)}{\lambda \cdot (N - E(M))}$$

Figure 5.3 illustrates the relationship between the mean response time and the number of processors, N, and the number of critical sections S, for fixed λ, L, and T for the finite source model.

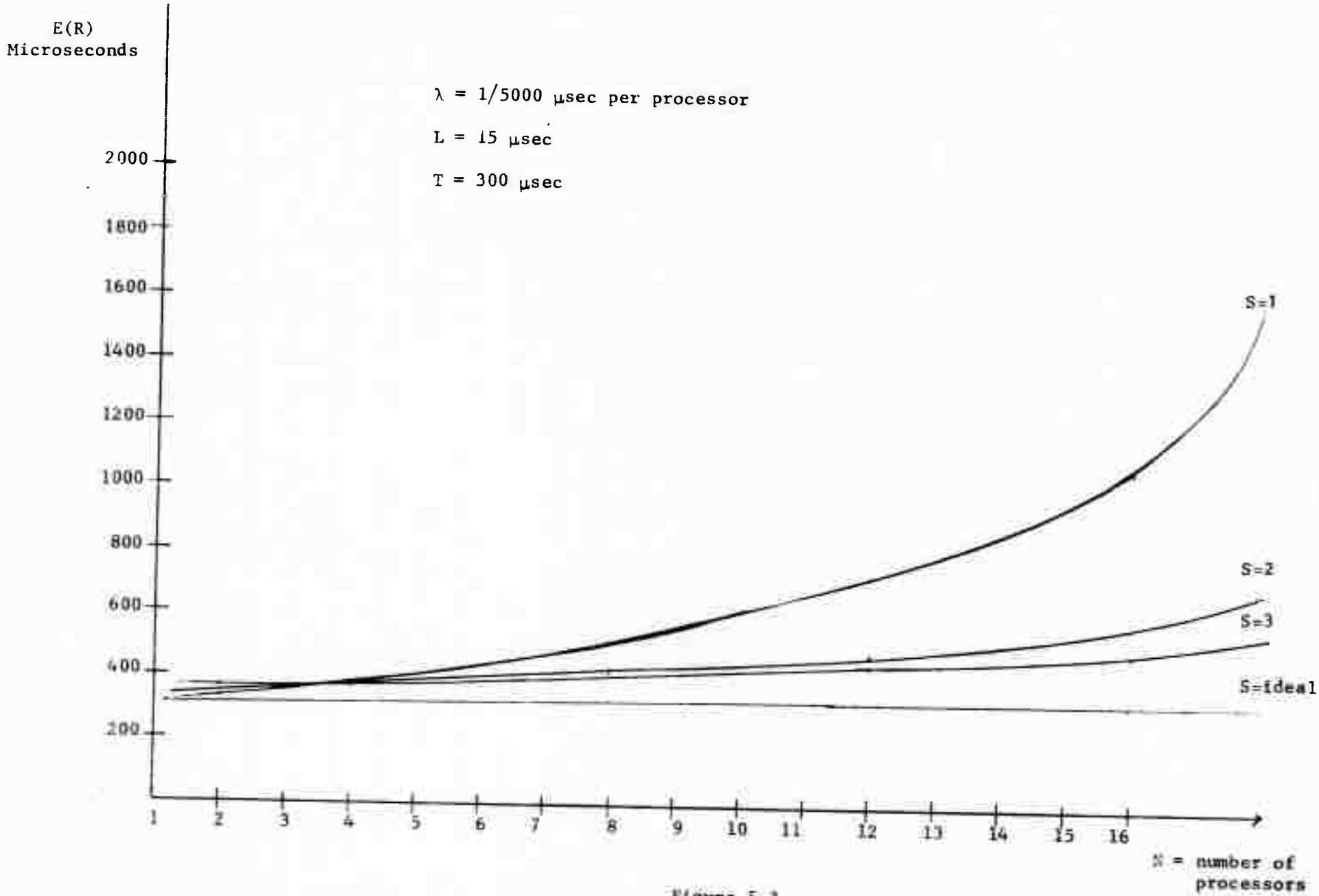


Figure 5.3
Mean Response Time for Scheduling Operations - MOD2

A lower bound for mean response time of scheduling requests may be obtained using an idealized state dependent service rate model based on the development of Section 2.5. In this model, the system is capable of dynamically reconfiguring the structure of the shared data base in the following way. Let m be the number of processors requiring access to the scheduling information at time t. The ideal system would provide m critical sections at time t. Whenever a processor completes its scheduling activity, or whenever a processor arrives with a new request, the accessing program will instantaneously change the structure of the data base to provide exactly the same number of critical sections as processors which need access to this data. In addition, all m processors will always have access to some data so that they will never be idle while attempting to use the shared scheduling information. Although this scheme is obviously impractical, it does lead to a lower bound for response time, and one may see how close to this bound a practical implementation may get.

Since every processor may use scheduling information at once, the exponential service rate for each processor is $1/(T+m\cdot L)$, where m is the constantly changing number of processors demanding access to the common data. A processor will not have to wait for any other processor, but it will pay a dynamic overhead penalty with rate $m\cdot L$ time units where m changes with the system state. Let $P_m(t)$ be the probability that there are m processors in critical sections at time t. As in Section 2.5 one may write the following system of state equations.

$$(5.4)\quad P_0(t+\delta) = (1-N\cdot\lambda\cdot\delta)\cdot P_0(t) + (1-(N-1)\cdot\lambda\delta)\cdot\delta\cdot P_1(t)/(T+L) + o(\delta^2)$$

$$P_m(t) = (N-(m-1))\cdot\lambda\cdot\delta\ \{1-(m-1)\cdot\delta/(T+(m-1)\cdot L)\}\cdot P_{m-1}(t)$$
$$+ (1-(N-m)\cdot\lambda\cdot\delta)\cdot\{1-m\cdot\delta/(T+m\cdot L)\}\cdot P_m(t)$$
$$+ \{(m+1)\cdot\delta/(T+(m+1)\cdot L)\}\cdot\{1-(N-(m+1)\cdot\lambda\cdot\delta\}\cdot P_{m+1}(t) + o(\delta^2),$$
$$m=1,2,\ldots,N-1$$

$$P_N(t) = \lambda\cdot\delta\cdot\{1-(N-1)\cdot\delta/(T+(N-1)\cdot L)\}\cdot P_{N-1}(t)$$
$$+ \{1-N\cdot\delta/(T+N\cdot L)\}\ P_N(t) + o(\delta^2)$$

Following exactly the same procedure as in Section 2.5, one first collects terms and then takes the limit as $\delta \to 0$ to get $P'_m(t)$. In statistical equilibrium $\lim_{t\to\infty} P'_m(t) = 0$, and $\lim_{t\to\infty} P_m(t) = P_m$. The result of this manipulation is the following set of steady state equations.

$$(5.5)\quad P_m = P_0 \cdot \prod_{i=1}^{m} \lambda\cdot(N-(i-1))\cdot(T+i\cdot L)/i, \qquad m=1,2,\ldots,N$$
$$P_0 = 1 - \sum_{i=1}^{N} P_m$$

One may calculate the mean number of processors in the blocked state by solving the above set of equations and then using equation (2.63). Equation (2.70) relates the mean response time for a scheduling request to the mean number of blocked processors. On Figure 5.3, the line labeled "S = ideal" is the plot of mean response time for this idealized version of the critical section scheduling problem.

5.2.3 Discussion of Results

Both Figure 5.2 and Figure 5.3 illustrate the same types of performance changes with respect to changes in the number of processors and the number of critical sections. Mean response time increases with the number of processors. For a constant number of critical sections, S, the increase

in mean response time is approximately linear, with respect to N, until the system becomes congested. As N increases beyond this point, the slope of mean response time as a function of N grows larger with increasing N. This non-linear response time degradation as a function of the number of processors is more severe in the first model (because processors continue to submit requests while blocked) than in the second, but it is evident in both.

The addition of one more critical section significantly improves mean response, for higher values of N, in both models. The additional locking overhead, L, associated with each critical section degrades performance slightly for small values of N. At these low values of N, the arrival rate of requests is so low that the extra locking overhead is not compensated for by the potential parallel utilization of the S critical sections.

An interesting characteristic of these models is the large performance improvement achieved by the creation of one or two additional critical sections. Figure 5.3 demonstrates that when S=2, the response time improvement is about one half of what the idealized system could provide. The improvement is greater for higher arrival rates of scheduling requests. The slight response time degradation for low arrival rates indicates that an efficient design would be the implementation of a few (S=2, or 3) critical sections. This choice would create an effective safety valve. Whenever the load would increase, parallel access to the data would occur and the shared scheduling information would not become a bottleneck. The overhead penalty at low arrival rates is in the neighborhood of only five percent and the improvement at higher request rates is approximately fifty percent.

All conclusions drawn from a model like this depend on the values of the input parameters. The variables isolated for this model are: λ, the mean arrival rate of scheduling requests from each processor; N, the number of processors; S, the number of critical sections; T, the mean time required to perform a scheduling operation; and L, the mean time needed to lock and unlock a critical section. The system designers can determine the values of T and L from code requirements and the speed of the processors, and they know the range over which N may vary. However, one needs an estimate of λ to determine the best value of S. Unfortunately, this parameter is very hard to estimate before the system has been built.

This basic dilemma is at the heart of many crucial design decisions. One needs the value of an important parameter to make a good implementation decision, but that value can only be estimated with a reasonable degree of confidence after the system has been in operation for some time. A good analytic model can be an important tool in such decisions. The nature of system response over a wide range of possible values may be easily studied. These sensitivity studies may lead directly to a solution, or they may help to plan a strategy for future experimentation and performance evaluations.

For example, the model of scheduling activity in C.mmp indicates that when the number of processors is less than four or five, and scheduling requests arrive with a mean interarrival time of five milliseconds (λ=1/5000) from each processor, a single critical section is all that is needed. As one adds more processors, or alternatively, if the rate of scheduling requests is much greater than this estimate, then one or two additional critical sections will improve performance significantly. Since both the

overhead penalties at low request rates, and the implementation costs, are small for these additional critical sections, the designers at Carnegie-Mellon chose a multiple locking strategy for C.mmp.

5.3 PERFORMANCE IMPROVEMENT ANALYSIS

Consider a time sharing system similar to the IBM 360/67. This system, using the operating system TSS/360, was the basis for the simulation presented in Section 3.5 and it was used for the empirical investigations reported in Section 4.2. Many of the powerful features of this system lead to high overhead costs. The virtual memory design gives every user a working space for programs and data that is much larger than the amount of high speed core memory in the system. The operating system manages this virtual memory so that users do not have to worry about storage allocation problems. When a user wants some information that is not currently stored in the high speed memory the system will automatically retrieve it from secondary storage. All information in the system is divided into blocks called pages. A page fault occurs when a program needs access to a page that is not currently in high speed memory.

Each page fault causes both an overhead operation, due to the many bookeeping functions that must be performed, and a request to the input/output (I/O) subsystem to retrieve the new page. Quite often a page already in memory must be placed in external storage to make room for the new page. The page fault rate is related to the amount of high speed core allocated to each active user. As each user's core allocation becomes smaller he will generate more page faults. Computers with a virtual memory often have been observed to enter a mode of operation, commonly

called thrashing,in which each user is allocated so little core memory that he generates a page fault very soon after gaining control of the central processor. Then the combined page request rate from all users may exceed the capacity of the I/O subsystem. When thrashing begins efficiency drops very quickly due to the combined effects of page fault overhead and the I/O system bottleneck.[1] A common cause of thrashing is allowing too many active programs to be squeezed into core memory with the result that none has enough of this resource.

A manager of this type of time-shared system can improve system performance in a number of ways. This section will illustrate how the models of Chapter 2 may aid in a study of alternative performance improvement plans. For example, changes in technology, or in operating budgets, may make it feasible either to increase the processing capacity of the central processor (C,measured in instructions executed per unit time) or to increase the total amount of core memory available. One may want to combine both methods. The performance measure used here will be the mean response time experienced by the users. The hypothetical system will have characteristics that are useful for illustrative purposes, and will not be representative of any particular implementation.

Increasing C, the instruction rate of the computer, will benefit users because their tasks will take less processing time. A common way of increasing C is to improve the performance of the memory in the system. For example, the 360/67 at Carnegie-Mellon was configured with a large amount of core memory, called LCS (Large Capacity Store), having a relatively slow cycle time. Improvements in memory technology made it possible to

[1] See Denning (1968) or (1970) for good discussion of thrashing.

increase the speed of similar memories by a factor of approximately three. If the number of active users is kept below the number that causes thrashing, what improvement may be expected in mean response time if C is increased by a factor of two or three and all other parameters remain constant? An increase in C would be of little help if the system were operating under conditions where thrashing could occur frequently.

Increasing the amount of core memory available will help users because they will be able to compute for longer intervals, when they have control of the central processor, before causing a page fault. Overhead will decrease because there will be fewer page faults to process, and the paging demand placed on the I/O system will also decrease. How much will a two or three fold decrease in the number of page faults experienced by a task benefit mean response time if all other parameters remain the same? What effect would one observe from a combination of these two possible performance improvements?

The hypothetical system which will form the basis for the analysis will be a computer with the following basic characteristics:[2]

(a) the original speed of the central processor will be 250,000 instructions per second;

(b) tasks arrive at the system with a mean rate of 2 per second;

(c) each task originally will issue an average of 40 page faults per request;

(d) each request will be for an exponentially distributed number of instructions having a mean value of $v1 = 50{,}000$;

[2] These operating characteristics are illustrative representations of some current time-shared computers which make use of extended core storage devices and a virtual memory design.

(e) the overhead time to process a page fault will be a random variable with a mean of 5 milliseconds; (Since a page fault will occur with a mean rate of once every 50,000/40 = 1,250 instructions, or about once every 5 milliseconds, the system is spending approximately 50 percent of its time in paging overhead operations.)

(f) the time required to locate and transfer a page of information from external storage will be approximately 10 milliseconds.

In the following analysis three different types of possible performance improvements will be considered:

Case (a) Mean overhead time required to process a page fault will remain constant at $d1 = .005$ seconds per page fault regardless of the value of C, the effective speed of the central processor. C will increase from 250,000 to 550,000 instructions per second.

Case (b) The mean number of instructions required to process a page fault will remain constant at $d1 \cdot I = 1250$ instructions. Thus the mean time needed to process each page fault will be $(d1 \cdot I/C)$. C will increase from 250,000 to 550,000 instructions per second.

Case (c) C will remain constant at 250,000 instructions per second, but the mean number of page faults generated by each task will decrease from 40 per interaction to 18.2 per interaction (ℓ increases from .025 to .055).

The distribution of the number of instructions a task requests will remain the same, but tasks will compute for longer periods of time before generating a page fault.

Increasing the effective speed of the central processor may, or may not, have an effect on time required to perform overhead operations associated with a page fault. Case (a) represents system designs in which an increase in the instruction rate of core memory dedicated to users does not affect the speed of overhead activities associated with the resource allocation functions of the operating system. For example, the system could have a memory hierarchy in which the operating system used the fastest, and most expensive, memory in the system and user programs ran in slower core memory (LCS). Performance improvements in LCS would not affect the speed of overhead functions which make use of the high speed memory. Case (b) represents the situation in which the performance of all memory in the system is improved. Case (c) illustrates the effects of an addition of more memory to the system with a resulting decrease in paging activity.

5.3.1 TSMOD1 Analysis

Figure 2.1 illustrates the structure of TSMOD1. Paging overhead is treated explicitly in this model. Overhead will be a random variable having a mean of d1 and standard deviation of d1/5. Equation 2.24 presents the functional relationship between mean response time and the other system parameters. Let the expected value of V (the number of instructions required to complete a task's request) be v1, and the expected value of W (the number of instructions executed between page faults) be w1. If V has an exponential distribution, W will also have an exponential distribution

with mean $\ell \cdot v1$ and second moment $w2 = 2 \cdot (w1)^2$. The values of all parameters needed for the initial configuration of the model are:

$$C = 250{,}000 \text{ instructions/second}$$
$$\lambda = 2 \text{ tasks/second}$$
$$\ell = 1/40$$
$$v1 = 50{,}000$$
$$w1 = \ell \cdot v1$$
$$w2 = 2 \cdot (\ell \cdot v1)^2$$
$$d1 = \begin{cases} .005, & \text{case (a)} \\ 1250/C, & \text{case (b)} \end{cases}$$
$$d2 = (d1/5)^2 + (d1)^2$$

Note that the I/O system does not appear in this model. An implicit assumption associated with this structure is that the I/O system is not a bottleneck and will not significantly affect performance. Any delays associated with page transfers will degrade performance from that predicted by TSMOD1.

Figure 5.4 presents graphs of the performance increases which result from the three different strategies (a), (b), and (c). The first example of Section 2.3.3 indicated that when overhead is not present and when all other parameters were held constant, changes in ℓ did not cause changes in the mean response time. As more quantizing took place (ℓ decreased) the standard deviation of response increased, but the mean remained the same. Figure 2.2 illustrated the effects of a type (c) improvement in an overhead free environment. However, the overhead included in the model of this section has a large effect upon mean response as ℓ changes. Cutting the paging rate in half is equivalent to doubling the speed of core dedicated to users. A careful examination of equation (2.34) indicates that an increase in C produces a response time improvement similar to a

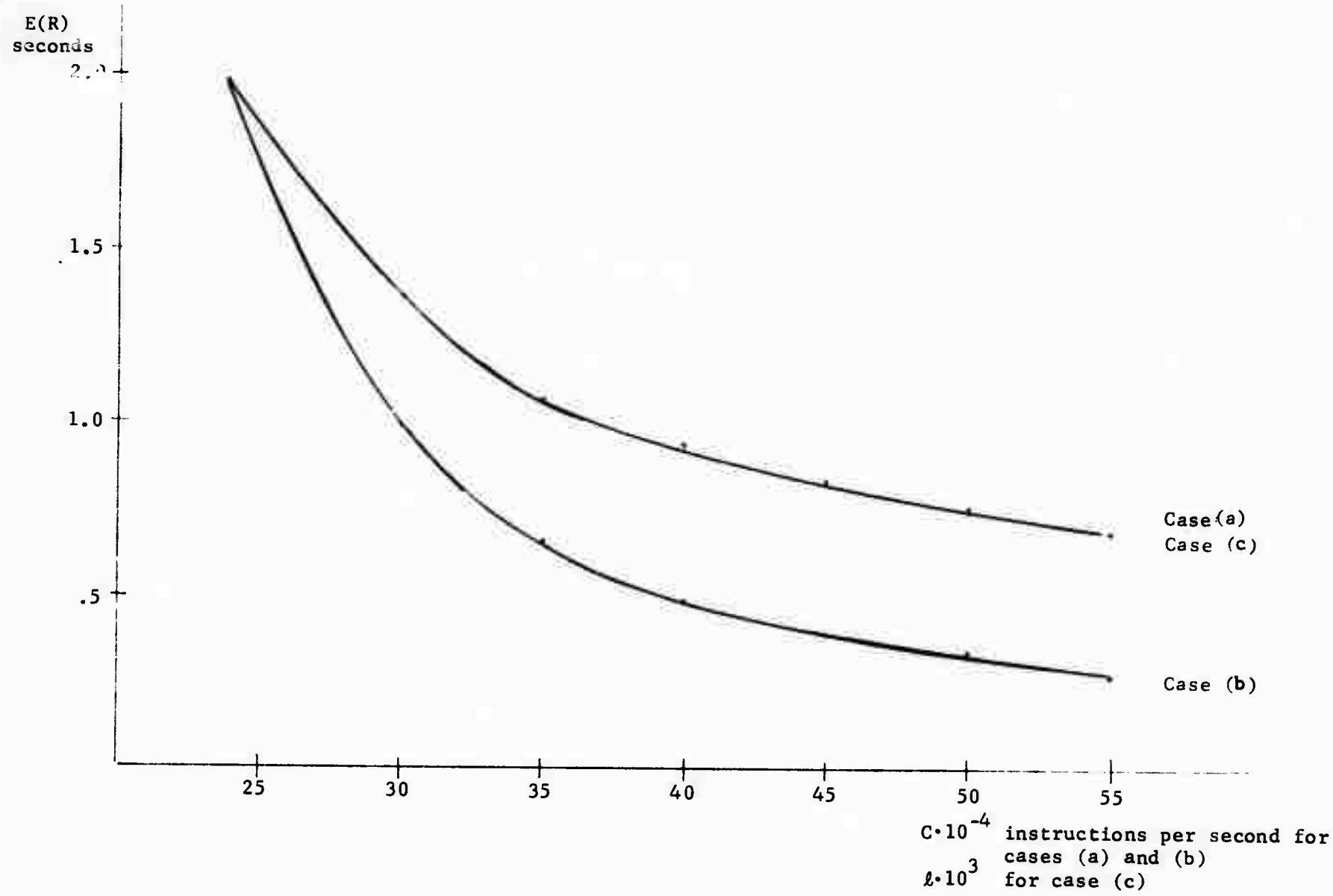


Figure 5.4

Mean Response as a Function cf C-TSMOD1

decrease in the number of instructions required to satisfy a task's request. A decrease in the paging rate (an increase in ℓ) produces an improvement similar to a reduction in overhead delay associated with the processing of page faults. Cases (a) and (c) produce almost exactly the same improvements. Case (b) reduces the load on the central processor by decreasing the times required to process both a task's request and the overhead associated with each page fault.

5.3.2 TSMOD2 Analysis

Figure 2.6 illustrates the structure of TSMOD2. An I/O subsystem in tandem with the central processor allows one to consider the effects of page transmission delays on mean response time. To keep the analytic formulation tractable, one must assume that the service times spent in both the central processor and the I/O system are exponentially distributed. The analysis of Section 2.4 did not include any provision for overhead degradation due to the processing of page faults.

One may introduce overhead into TSMOD2 in the following way. Let the service time spent in the central processor be an exponentially distributed random variable having a mean of $(v1/C + .005/\ell)$ seconds for case (a) and a mean of $(v1/C + 1250/(C\cdot\ell))$ seconds for case (b). As in the previous section the total number of instructions required to satisfy a task's request will have a mean value of v1. The mean time spent in the central processor during each quantum interval will be an exponentially distributed random variable with mean $((v1\cdot\ell)/C + .005)$ seconds for case (a) and mean $((v1\cdot\ell + 1250)/C)$ seconds for case (b). This formulation retains the essential features of a random quantum interval which is divided into an overhead segment and a processing segment.

Equation (2.55) presents the mean response time in TSMOD2 conditioned on the fact that a job requires v instructions. Since this is a linear function of v,one may remove the condition by replacing v with its expected value presented in the preceding paragraph. Let the time required to find, and then transfer, a page from external storage be an exponentially distributed random variable with mean w12 = .01 seconds. All parameters required for this model have now been specified.

Figure 5.5 is a graph of the performance increases achieved by using the three improvement plans. In this model an increase in C, the processing rate of the central computer, has no effect on the I/O subsystem. Case (b) is again better than case (a) because an increase in C reduces the overhead delay. But the best of the three strategies is case (c). A reduction in the paging rate achieved by the addition of more core reduces both overhead and the demand on the I/O system. For case (c) the system maintains a better overall balance, and neither subsystem becomes overly congested.

5.3.3 TSMOD3 Analysis

TSMOD3 is a finite source, processor-shared, model with an overhead loss which is a function of the state of the system. Figure 2.8 illustrates the structure of this configuration. Like TSMOD1, this model does not have an I/O subsystem and thus an implicit assumption in its use is that the I/O delay is not significant. Any delay in the I/O system will add to response times computed by this formulation.

The model does not explicitly consider any overhead associated with the processing of page faults. The state dependent overhead loss represents

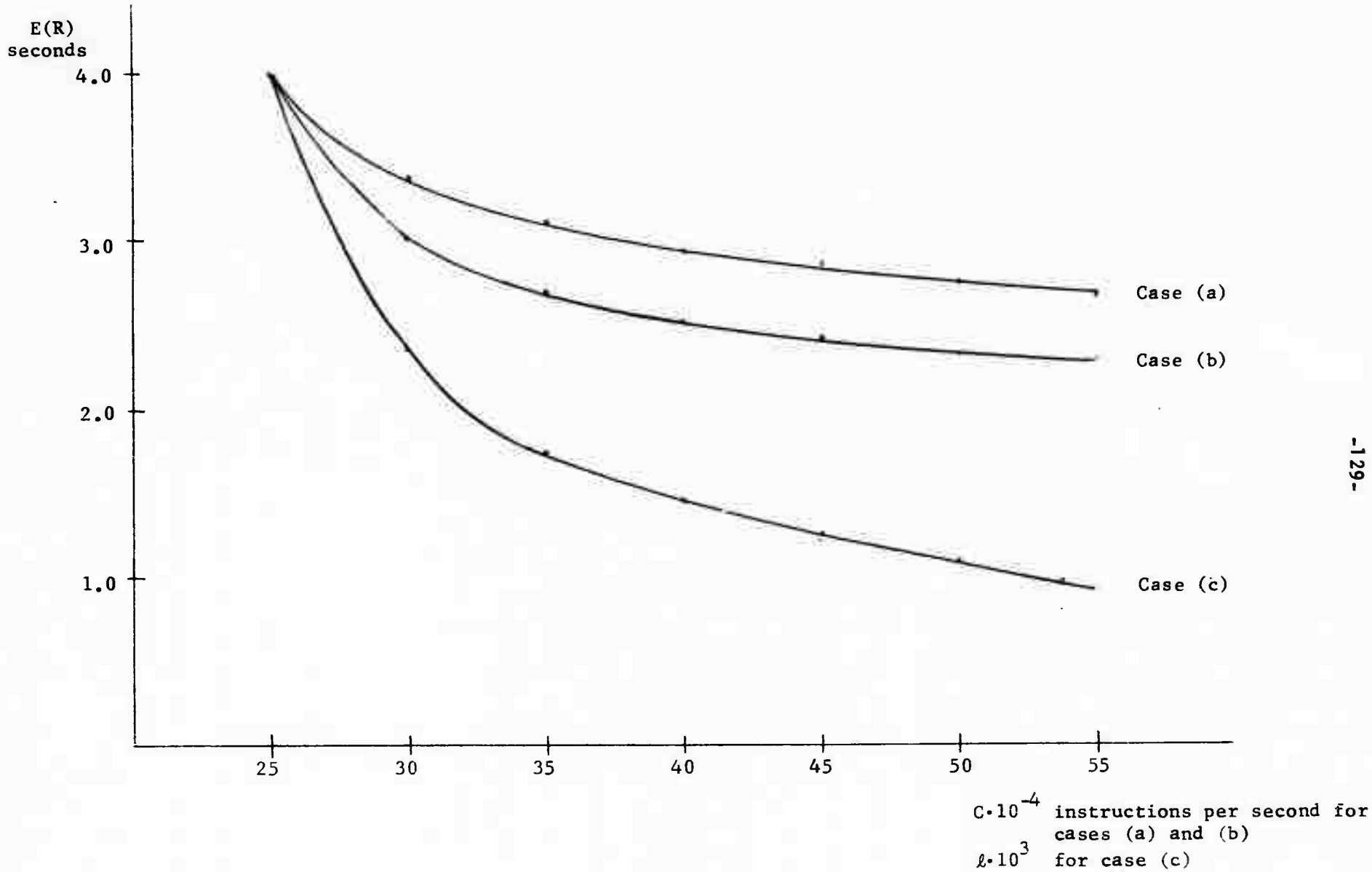


Figure 5.5

Mean Response as a Function of C-TSMOD2

general overhead degradation, but it is not a function of the paging rate. To apply the model to the situation of interest in this section one may take the same approach as in Section 5.3.2. The instantaneous exponential rate for each job in the processor-shared system was called $v \cdot C \cdot (1-f \cdot m)$ in Section 2.5. The number of tasks in the processor at each instant of time is m. Let $v \cdot C$ be equal to $\ell \cdot C/(v1 \cdot \ell + .005 \cdot C)$ for case (a) and $\ell \cdot C/(v1 \cdot \ell + 1250)$ for case (b). Since the mean of an expenential process with rate $v \cdot C$ is $1/(v \cdot C)$ this formulation leads to the same expected processing times as the previous derivation. This model also retains the essential characteristic of a random quantum interval which is divided into an overhead and a processing segment. The state dependent loss factor $(1-f \cdot m)$ will be applied as it was in Section 2.5.

Let N, the number of terminals making requests upon the system, be 40 and let the time between the completion of one request and the submission of the next be an exponentially distributed random variable with a mean of 20 seconds ($\lambda = 1/20$). Let the overhead loss fraction, f, be .02. All parameters required for TSMOD3 have now been specified and one may examine cases (a), (b), and (c). Figure 5.6 is a graph of the performance increases achieved by using these three plans. An examination of the equations of Section 2.5 indicates that for cases (a) and (c) an increase in the processing rate, C, is equivalent to a reduction in the paging rate. Thus the curves for cases (a) and (c) coincide. Case (b) achieves a better level of performance because an increase in C has a direct effect on the overhead delay associated with each page fault.

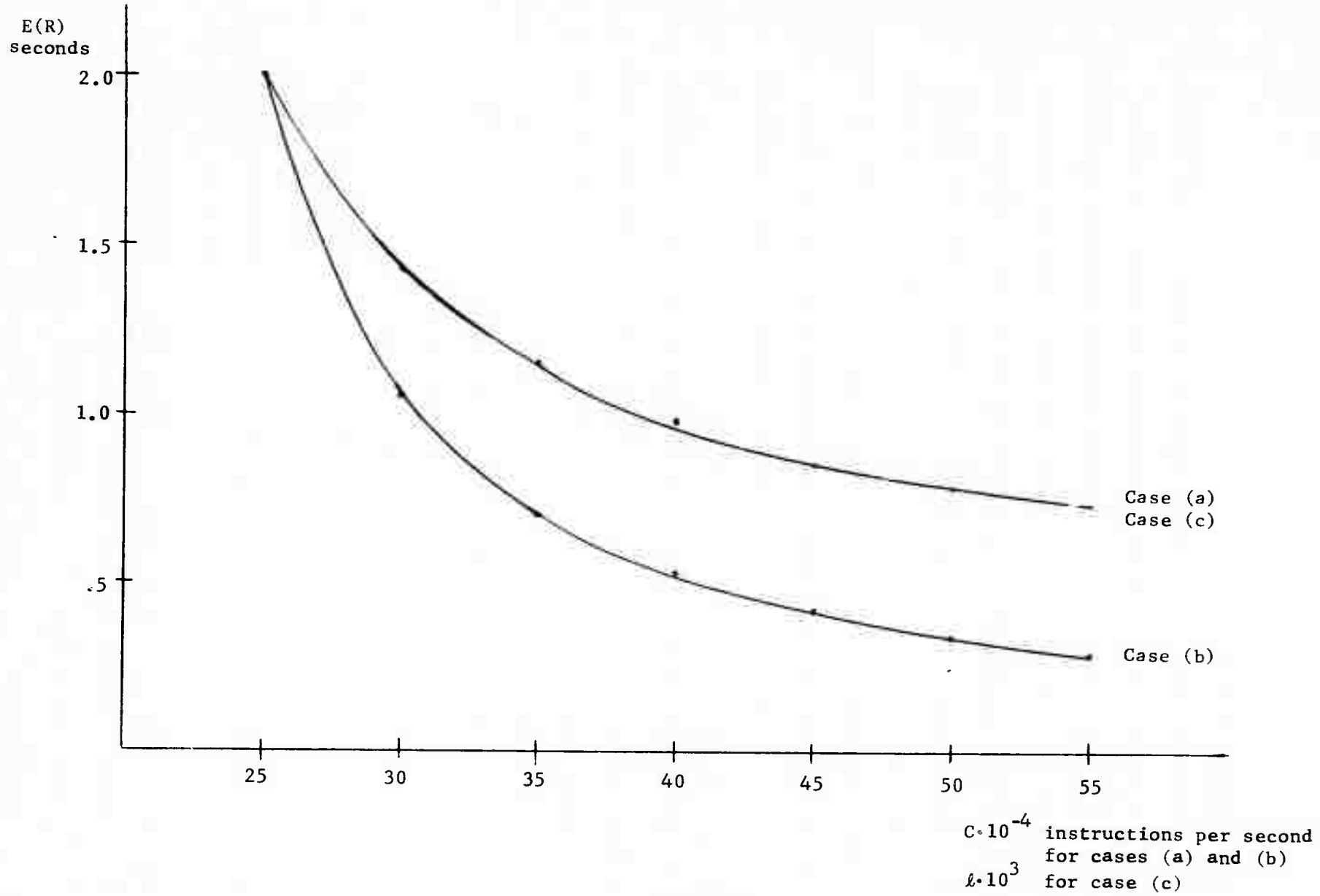


Figure 5.6

Mean Response as a Function of C-TSMOD3

5.3.4 Discussion of Results

Each of the three models used to study the hypothetical performance improvement problem concentrates on a few important features of time-shared computing systems. In the previous sections each model was modified so that it could be applied to a problem for which it was not specifically designed. For example, TSMOD1 and TSMOD3 do not include subsystems which can represent input/output activities. TSMOD3 is the only formulation which considers the effects of both a finite user population and an overhead degradation which is a function of system state. TSMOD2 and TSMOD3 both require that service requests be exponentially distributed random variables, and neither of these two models includes an explicit mechanism for studying the effects of paging overhead.

By carefully redefining some important parameters one may apply all of these models to the performance improvement problem. Each model allows the analyst to focus on a different aspect of system design. The results clearly show that performance may be significantly improved by either increasing the processing rate or decreasing the paging rate. All three graphs also indicate that the performance improvement curves level out after an initial interval of a higher rate of change.

All of the models point to additional studies which should be made. For example, none includes a functional relationship between the size of core and the paging rate. All of the models predict significant performance improvements if one can lower the paging rate, but the amount of additional core which would be required to cut the paging rate in half (by doubling ℓ) is beyond the scope of these analytic models. A simulation such as the one presented in Section 3.5 can help with this problem. In addition,

a carefully designed experiment on the current configuration could help quantify this relationship.

The following list illustrates some (of the many) additional considerations which would be required prior to an actual performance improvement decision: the relative costs of more versus faster memory; comparative reliabilitites of different memories; maintenance problems; vendor compatability; purchasing and leasing agreements; interface problems with other system components; predictions of future usage patterns and technological improvements; changeover costs; other subsystem improvements. The use of analytic models as a performance analysis tool can help reduce uncertainty in some of these dimensions and thereby improve the decision making process.

5.4 DYNAMIC SYSTEM CONTROL

All of the models applied to the performance improvement problem of the previous section show that response time will increase with the paging rate. As the number of users competing for core in a virtual memory system grows the paging rate increases. At the end of a quantum interval, systems which do not have a virtual memory structure must often save, in external storage, that portion of a user's program and data which is currently in core memory. The saving of one core image in external storage and the retrieval of a different one is usually called swapping. Swapping need not occur at the end of every quantum because core may be large enough to hold several tasks at one time. As the load increases, and more users demand service, the swapping rate, like the paging rate in a virtual memory design, will increase. The overhead associated with paging and swapping will cause response time degradation.

Figures 2.3, 2.6, 2.7, and 2.9 show that as the arrival rate of requests grows, expected response time increases in a nonlinear manner. This degradation, as a function of the arrival rate, occurs even when there is no overhead associated with swapping or the processing of page faults. With overhead present the effects are magnified. The phenomenon of thrashing, which was described in the previous section, is an extreme example of what can happen when overloading occurs.

Quick response to short requests is a major goal of time-shared computing systems. To maintain a reasonable level of response all such systems must limit the input rate of requests. For example, at Carnegie-Mellon a Logon Priority system limits the number of interactive terminal users to a pre-set limit. If someone tries to join the system and the number of people currently logged on is equal to the limit, the new user is denied access until the system can force one of the active users to leave. The algorithm which makes the decision,about which job should be forced, considers factors such as pre-assigned priorities and the length of time each of the current users has been connected to the system. The algorithm chooses a user to be forced from the system and then notifies him that he must leave within the next two minutes or be automatically terminated. If the algorithm is unable to find a user who meets all criteria for automatic termination, the new user is denied access. Once a user has been allowed to logon to the system he is guaranteed a minimum length of time during which he may use the computer.

McCredie (1967), Wulf (1969), Wilkes (1971), Mills (1971), and others have suggested that dynamic load adjustment procedures be used to control the performance of computing systems. CTSS, a time-shared system developed

at MIT, has such an automatic load leveling capability built into the operating system. Wilkes (1971) presents an analysis of the stability of such a system and Mills (1971) describes the algorithm in use at MIT. The objective of this section is to illustrate how analytic models such as those developed in Chapter 2 may be used as an integral part of such a control system. Clearly a model for this purpose should not require a great deal of computer time to solve, or any gains resulting from the use of the model could be lost in the extra overhead required to support the control algorithm.

5.4.1 The System

Figure 5.7 illustrates the structure of the combined computer and user subsystems. Of all potential users, only a fraction will want to interact with the computer at any particular time. The system will determine how many active users, N, will be able to establish a connection (logon) and then use the computer. The procedure which performs this control function will be called the Terminal Allocation Algorithm (TAA). Appendix E contains a listing of the SIMULA program used to investigate a few (of the many possible) different versions of a Terminal Allocation Algorithm.

The global structure of the program is illustrated in Figure 5.7. The data gathering facilities and the central processor and input/output subsystems are slightly modified versions of the simulation used to study TSMOD2 in Section 3.4. The changes from the previous model are the additions of multiple input queues (based on the priority of a job) for each subsystem, and the modification of the overhead portion of each quantum

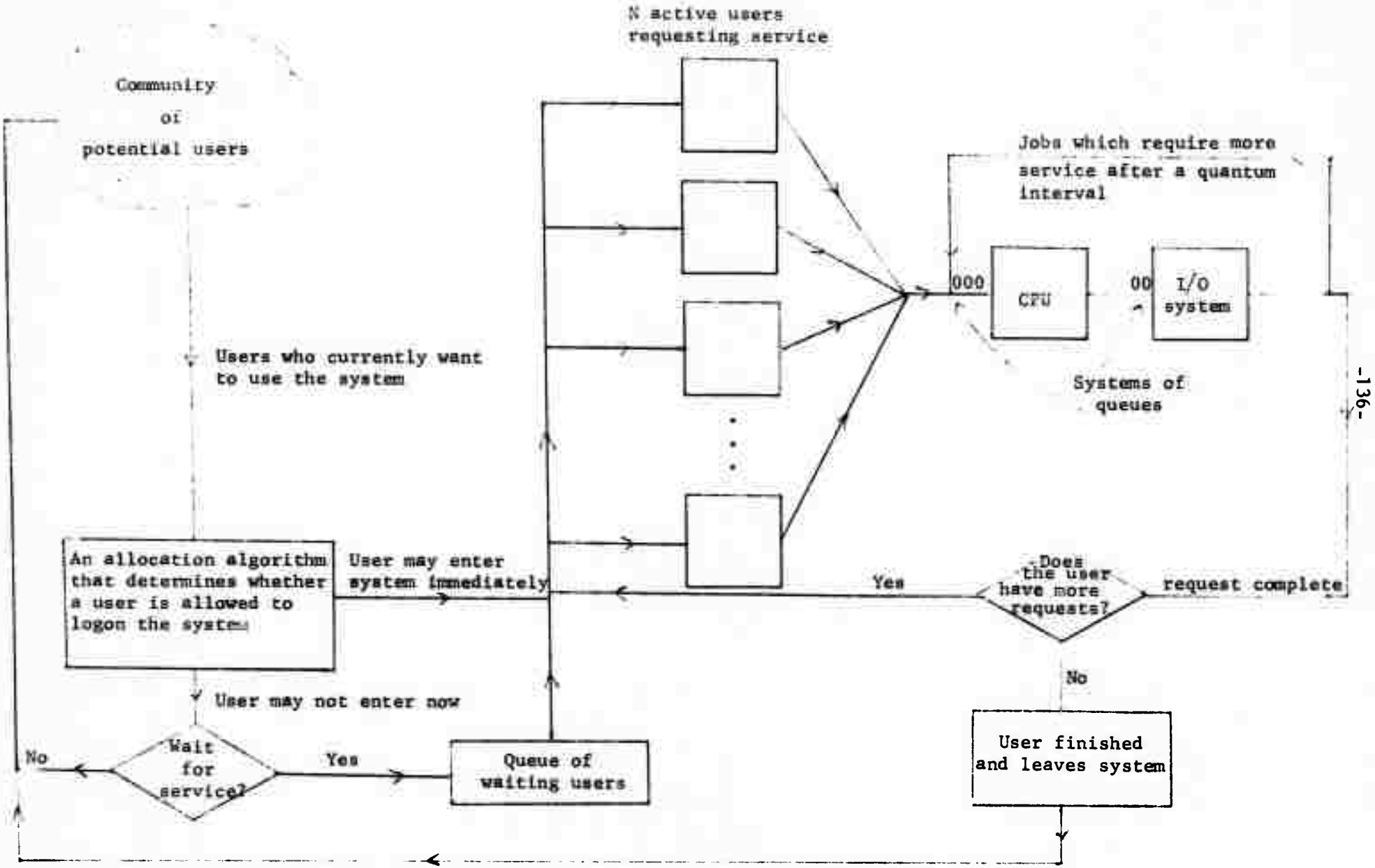


Figure 5.7

Structure of Allocation Model

to include a constant part and a portion which is proportional to the number of tasks waiting for service. Users at terminals are represented by a SIMULA activity, called USER, which creates requests and inputs them to the computing subsystem. After each request has been completed the user gathers statistics about his response time and then either creates a new request or leaves the system when all requests have been satisfied. Each user is inactive while waiting for the computer to finish a request. Another SIMULA activity, called the GENERATOR, creates users who try to gain access to the system. The Terminal Allocation Algorithm decides whether or not a new user may logon to the system.

5.4.2 TAA1

The first Terminal Allocation Algorithm evaluated is the default, or null, algorithm. Every job requesting service is admitted to the system regardless of the current load. After an intitialization period during which 20 users submit a total of 100 requests, new users logon to the system at a rate slightly greater than users leave the system. All users have the same statistical properties and the same priority.

The line labeled TAA1 on Figure 5.8 shows an increase in average response times as a function of time. As the number of users in the system increases, average response time increases in a nonlinear fashion.

5.4.3 TAA2

Using knowledge gained from the behavior of models like those of Chapters 2 and 3, one may formulate a simple but effective Terminal Allocation Algorithm based upon a limit to the number of people using the system. The only value needed by the algorithm is the number of users

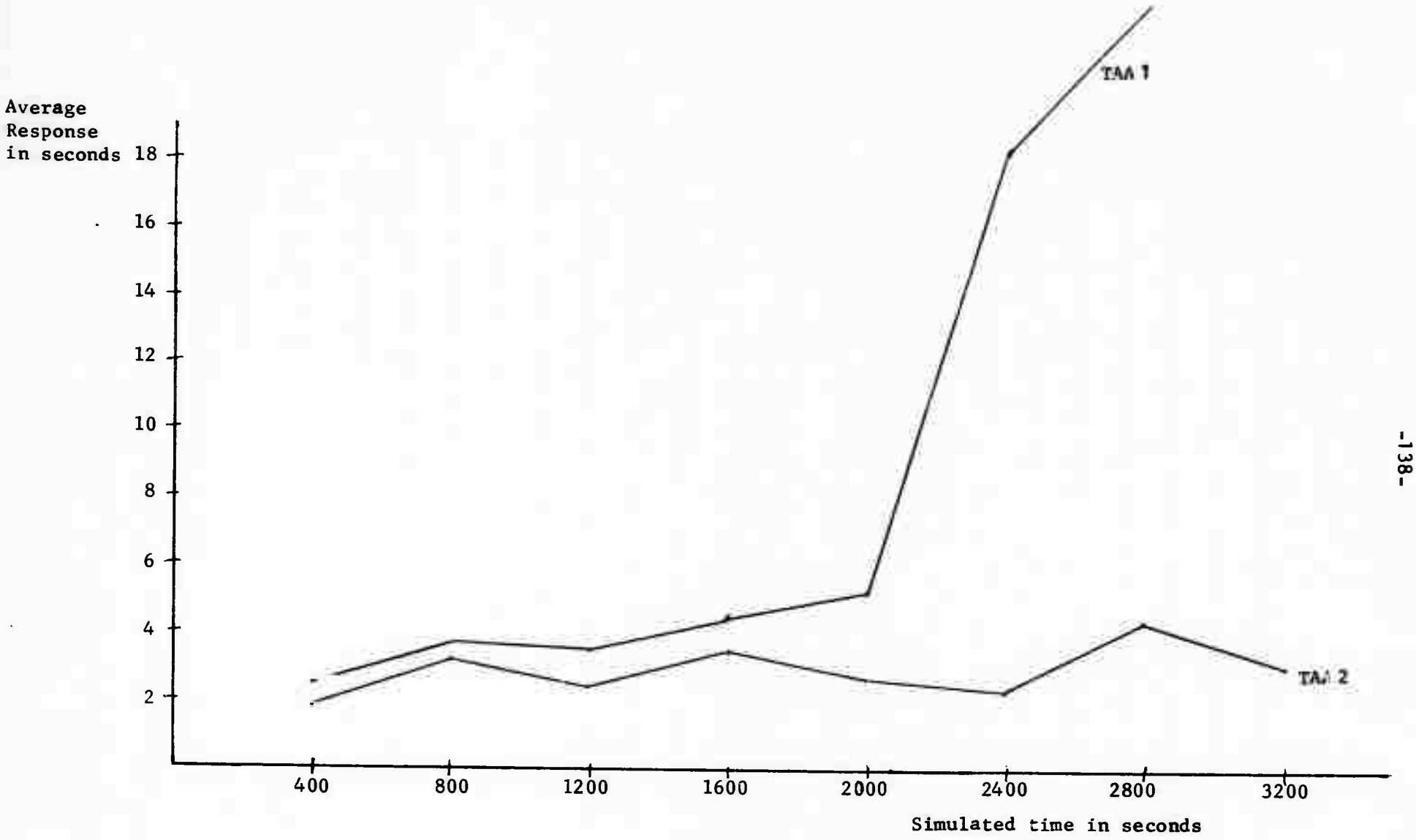


Figure 5.8

Average Response as a Function of Time for TAA1 and TAA2

currently logged onto the system. If the number is greater than the control limit any new user is denied access to the system. Whenever a current user finishes work and leaves the system a new user is allowed to logon if one is still waiting.

The line labeled TAA2 on Figure 5.8 shows that the simple strategy of controlling the input rate to the system will keep response times within design limits. Any idle time generated by limiting the maximum number of users may be allocated to lower priority tasks which may be interrupted with the arrival of more higher priority work.

5.4.4 TAA3

The third Terminal Allocation Algorithm is based upon the type of load balancing mechanism in use on the CTSS system at MIT. The control algorithm of TAA3 samples the state of the system periodically, and dynamically adjusts the maximum number of users who are allowed to logon to the system. Between sampling intervals the algorithm acts exactly like TAA2. If a user tries to logon, and the number of users already active is greater than or equal to the control limit, the new user is denied access to the system.

The philosophy underlying dynamic adjustment of the maximum number of users is based upon the observation that users have widely varying modes of interactive computer usage patterns. The parameters describing usage and input rates vary with time. If these parameters were stationary with respect to time one could choose a value for the maximum number of active users, Nmax, and never change it. If all Nmax users are performing editing types of functions the central processor may be underloaded, and the system

could support more users. Alternatively, if all Nmax users are compiling and running large programs the system could become overloaded. A control system that monitors the actual state of the system and balances the load accordingly can increase the number of users when there is excess capacity, and can reduce the input rate of requests when overloading occurs.

At each sampling instant TAA3 estimates values of the input rate of requests from the active users, the sizes of the queues in front of the central processor and the I/O system, and the rate at which tasks request service from the I/O subsystem. Using these values, TAA3 estimates both the number of tasks currently being processed by the computer and their characteristics. Using TSMOD2, the tandem queueing model developed in Section 2.4, the control algorithm then computes what effects the addition of another active user would have on the state of the system. If the predicted value of the system state is within the control range, the maximum number of users allowed to logon is increased by unity. If the addition of one more active terminal causes the predicted system state to exceed the control limit, the maximum number of active users is not changed. If the measurements indicate that system state has already exceeded the control values, the maximum number of active users is decreased by unity. However, in this implementation, no users are forced to leave the system before their session is complete.

It is a well known fact from the field of control theory that control algorithms, such as the one outlined above, are subject to severe instabilities. Since both the input and service functions are stochastic processes all of the parameter estimates are random variables subject to statistical fluctuations. Wilkes (1971) examines the stability of a simplified version

of the previously described dynamic control algorithm and demonstrates that instabilities are possible in practical situations. In addition to the variance of the parameter estimates, the time delays between changes in the control variables and their resulting effects make the proper choice of a control strategy a difficult problem. The theoretical treatment of this control problem, as applied to computer systems, is an area of future work which is not treated in the present development.

The particular version of TAA3 used in the following simulation is listed in Appendix E as activity ESTIMATOR. A common method of estimating non-stationary random variables is to form an estimate at each review period which is based on a combination of the past information and the current observations. TAA3 uses the following exponentially smoothed estimator for all state variables:

$$(5.6) \quad \bar{X}_{k+1} = (1-\alpha) \cdot \bar{X}_k + \alpha \cdot S_{k,k+1}$$

$\bar{X}_k$ is the value of the estimator at the end of the k^{th} period, and $S_{k,k+1}$ is the sample observations which occur during period k+1. In the simulation, $\alpha = 1/3$ and the time period between each sample was 25 seconds. These values were found by trial and error to smooth the statistical fluctuations of the process, and to track changes in the parameters of the users.

For this simulation each new user draws the parameters, which describe his usage mode, from probability distribution functions. Two parameters characterize user behavior:

(1) the mean value of the service request

(2) the mean time a user spends in the "think" state between the completion of one task and the submittal of the next.

Each user's parameters remain constant over the duration of his terminal session, but each request and each thinking interval are random variables drawn from a distribution characterized by the constant parameters. Thus the total user load will change as the parameters change.

5.4.5 Discussion of Results

Table 5.1 presents the results of a simulation of this environment using TAA2 with the maximum number of users set at 30. Table 5.2 presents the results of a simulation of the same environment using TAA3, the dynamic control algorithm. $\bar{S}$ is the estimated average number of tasks being processed by both the central processor and the I/O system. TAA3 tried to control this value at 3.5. The average number of interactions processed during each reporting period of 400 simulated seconds is approximately the same although TAA3 is slightly higher (381 versus 392). The average number of tasks in the system and their average response times are significantly larger for TAA2 than for TAA3 (4.56 versus 3.54 and 4.83 versus 3.84). $\bar{P}$ is the average percentage of idle time spent by the central processor. TAA2 had significantly less idle time (.12 versus .19) than TAA3. This time is not wasted since it can be used for background tasks of lower priority.

By controlling the average number of tasks demanding service, TAA3 is able to significantly improve system performance. TAA3 uses TSMOD2, the tandem queueing model, to evaluate the effects of proposed changes in the control variables. The magnitude of the potential performance improvements indicate that future investigations should examine the problem of designing an allocation algorithm which is optimal with respect to stability and performance objectives. The goals of this section were to illustrate how such an algorithm can use analytic models, such as those developed in Chapter 2, and to investigate what orders of magnitude of performance improvement one may expect from the implementation of static and dynamic control policies.

time/400	Number of interactions	$\bar{S}$	$\bar{R}$	$\bar{P}$
1	408	2.46	2.32	.28
2	484	4.07	3.22	.13
3	401	3.72	3.80	.13
4	363	4.86	5.34	.09
5	431	6.80	6.08	.02
6	386	7.68	7.64	.02
7	396	8.22	8.54	.03
8	393	5.11	5.47	.07
9	359	4.90	5.64	.09
10	340	3.1	3.71	.18
11	343	3.41	3.82	.17
12	428	4.10	3.89	.12
13	357	4.20	4.63	.12
14	321	3.30	3.76	.23
15	296	2.78	4.16	.29
16	376	4.21	4.80	.12
17	355	3.97	4.41	.10
18	426	3.73	3.54	.12
19	322	5.94	7.25	.03
20	416	4.66	4.53	.08
$\bar{X}$	381	4.56	4.83	.12
s^2	2064	2.40	2.48	.006
s	45	1.55	1.57	.076
$(s^2/20)^{\frac{1}{2}}$	10.16	.35	.35	.017

Table 5.1

Results of Simulation of TAA2

time/400	Number of interactions	$\bar{S}$	$\bar{R}$	$\bar{P}$
1	412	2.51	2.30	.28
2	417	3.39	3.34	.19
3	466	3.09	2.69	.22
4	558	4.50	3.10	.12
5	372	3.79	3.77	.18
6	148	3.87	7.00	.13
7	386	4.36	4.08	.12
8	234	3.12	6.09	.19
9	311	3.46	4.42	.17
10	467	1.83	1.51	.41
11	502	5.14	3.47	.14
12	295	5.50	8.64	.06
13	296	3.27	4.40	.21
14	338	3.11	3.18	.21
15	325	2.71	3.39	.26
16	492	4.56	3.59	.12
17	411	3.11	2.99	.18
18	452	2.76	2.49	.26
19	465	3.70	3.11	.17
20	352	3.04	3.53	.24
$\bar{X}$	392	3.54	3.84	.19
s^2	7815	.82	2.77	.006
s	88	.91	1.67	.076
$(s^2/20)^{\frac{1}{2}}$	19.77	.20	.37	.017

Table 5.2

Results of Simulation of TAA3

5.5 CONCLUSIONS AND FUTURE WORK

Chapter 1 contained a discussion of the role of analytic models as one of the techniques useful in the analysis of computing systems. The interactions of the analytic approach with simulations and empirical studies were explored, and the work of the following chapters was placed in context with other studies.

Chapter 2 presented the derivation of a number of performance measures of models of time-shared computer systems. Simplifying approximations, developed for some of these results, were compared to the exact expressions. Each model focussed upon a different feature of current time-shared implementations. Each of the models is easy to understand and use because the results are not complicated to compute, and because the structure of the model is obvious to the user. Since relatively few (five to nine) variables specify each system, the models are easy to control in sensitivity studies.

Chapters 3 and 4 presented evidence that the models are robust in the sense that the behavior they predict is observed in a wide range of related systems (both simulated and actual). The previous sections of Chapter 5 illustrated how such models may be used by designers and managers of computer systems. The example of Section 5.2 is a case study of how system implementors were guided in an important decision concerning the design of part of the operating system of C.mmp, the Carnegie-Mellon multi-mini-processor. Thus the models meet a number of the criteria, stated by John Little in Chapter 1, required of models which are to be used by designers and managers.

One of the characteristics of much of the current literature dealing with analytic models of computer systems is that it lags developments of actual systems. Increased communication among those who design systems and operations research specialists who create new models would help to reduce this time delay. Many currently available models are directly applicable to present design problems. For example, the area of the allocation and scheduling of resources in networks of computer systems is one in which models, such as those of Chapter 2, could be helpful.

Analytic models are greatly simplified abstractions of real computing systems. Therefore it seems appropriate for future model builders to concentrate more on simplifications which lead to useful approximations of important system problems, then on exact, but very complicated, solutions to minor modifications of existing structures. Naturally such approximations must be carefully studied to determine their domain of applicability.

Analytic modelling is only one of many techniques available to those who want to analyze, measure, improve, and create better computing systems. One of the goals of this report is to help place this approach to system modelling into perspective as an important tool, not a panacea, for computer scientists.

APPENDIX A

DERIVATIONS

This appendix contains a number of derivations which were referred to in the body of the report, but did not fit in any previous major line of development. The first section presents helpful, but non-standard, methods of calculating the first and second moments of non-negative random variables. These results are used in Section A2 which contains the correct expression for the second moment of the truncated exponential quantum interval of the Coffman and Kleinrock (1968) model, and in A3 which contains a derivation of the distribution and moments of Q_r, the remaining service time distribution of a quantum in progress when a new job arrives. Section A4 presents a solution to the Poisson source, exponential service with constant overhead associated with each quantum, model analyzed by Adiri and Avi-Itzhak (1969) and Rasch (1970). The solution method is the one used in Chapter 2, and the results are identical to those obtained by Adiri and Avi-Itzhak who used more complicated transform methods.

A1. THE FIRST TWO MOMENTS OF A NON-NEGATIVE RANDOM VARIABLE

The well known technique of integration by parts forms the basis for the following analysis. Integrals may often be simplified by application of equation (A.1).

$$(A.1) \quad \int_a^b u\,dV = uV\Big|_a^b - \int_a^b V\,du$$

If X is a non-negative random variable having finite first and second moments, E(X) and $E(X^2)$, and a distribution function $F_X(t)$, then:

(A.2) $E(X) = \int_0^\infty (1-F_X(t))dt$

(A.3) $E(X^2) = 2\int_0^\infty t(1-F_X(t))dt$

Proof

Apply (A.1) to (A.2) with the following substitutions:

$u = 1-F_X(t)$ $\qquad$ $du = -dF_X(t)$

$dV = dt$ $\qquad$ $V = t$

(A.4) $\int_0^\infty (1-F_X(t))dt = t\cdot(1-F_X(t))\Big|_0^\infty - \int_0^\infty t(-dF_X(t)) = \int_0^\infty t dF_X(t)) = E(X)$

Apply (A.1) to (A.3) with the following substitutions:

$u = 1-F_X(t)$ $\qquad$ $du = -dF_X(t)$

$dV = 1t\ dt$ $\qquad$ $V = t^2$

(A.5) $2\cdot\int_0^\infty t\cdot(1-F_X(t))dt = t^2\cdot(1-F_X(t))\Big|_0^\infty - \int_0^\infty t^2\cdot(-dF_X(t)) = E(X^2)$

The term $t\cdot(1-F_X(t))\Big|_0^\infty$ is zero because $t\cdot(1-F_X(t)) \le \int_t^\infty y\cdot dF_X(y)$

and $\lim_{t\to\infty} \int_t^\infty y dF_X(y) = 0$ since $E(X)$ exists. Similarly, $t^2\cdot(1-F_X(t))\Big|_0^\infty$

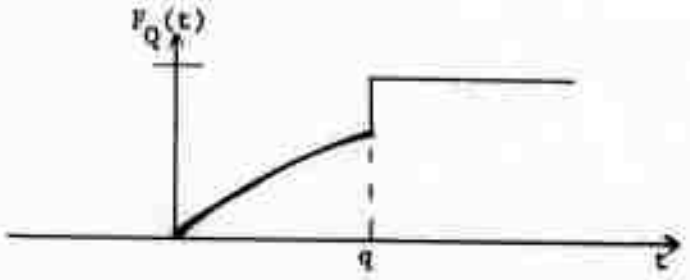
is zero because $t^2\cdot(1-F_X(t)) \le \int_t^\infty y^2 dF_X(y)$ and $\lim_{t\to\infty} \int_t^\infty y^2 dF_X(y) = 0$ since

$E(X^2)$ exists.

A2. THE SECOND MOMENT OF A TRUNCATED EXPONENTIAL QUANTUM INTERVAL

The second moment of a quantum interval is an important quantity in the Coffman and Kleinrock (1968) article. They assume that service quanta have the following distribution function:

$$(A.6)\quad F_Q(t) = \begin{cases} 0 & t < 0 \\ 1-e^{-ut} & 0 \le t < q \\ 1 & t \ge q \end{cases}$$



The second moment of Q may be easily calculated using equation (A.3).

$$(A.7)\quad E(Q^2) = 2\int_0^\infty t(1-F_Q(t))dt$$

$$= 2\int_0^q te^{-ut}dt + 2\int_{q^+}^\infty t\cdot 0dt = 2\int_0^q te^{-ut}dt$$

Apply (A.1) to (A.7) with the following substitutions:

$$u = t \qquad du = dt$$

$$dV = 2e^{-ut}dt \qquad V = \frac{-2e^{-ut}}{u}$$

$$E(Q^2) = \frac{-2te^{-ut}}{u}\Big|_0^q + \frac{2}{u}\int_0^q e^{-ut}dt$$

$$= \frac{-2qe^{-uq}}{u} + \frac{2}{u^2}(1-e^{-uq})$$

$$= \frac{2}{u^2} - \frac{e^{-uq}}{u^2}(2uq+2)$$

The final form of equation (A.7) corrects equation (16) of Coffman and Kleinrock in which they assert the following incorrect result:[1]

$$(A.8)\quad E(Q^2) = \frac{2}{u^2} - \frac{e^{-uq}}{u^2}(u^2q^2 + 2uq+2) \qquad \text{INCORRECT}$$

A3. REMAINING SERVICE TIME - Q_r

Given that a job from a Poisson source enters a system and finds a job being served, what is the distribution of Q_r, the time from the arrival of the new job until the one being served finishes? Section 2.2.4 contains a discussion of this quantity. Conway, Maxwell and Miller (1967), Chapter 8, pages 146-147, present a derivation of the properties of Q_r. Since this quantity is very important to the results of Chapter 2, and since Shemer (1967) did not realize that Q_r had a distribution different from other quanta, this section will present a derivation of the distribution of Q_r, its Laplace transform, and its moments. The following derivation differs substantially from Conway, Maxwell and Miller, but the results are the same.

Let Y be the elapsed time from the beginning of a service interval until a new task arrives. A well known result of renewal theory is that if a job finds another being served when it arrives, then:[2]

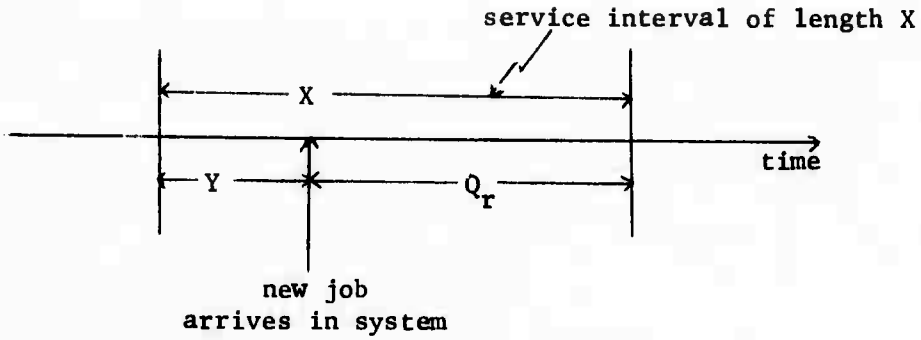
$$(A.9)\quad P(y \le Y \le y+dy) = \{(1-F_X(y))/E(X)\}dy, \quad y \ge 0$$

where $F_X(y)$ is the distribution function of a service interval.

---

[1] Coffman, E., and Kleinrock, L., "Feedback Queueing Models for Time-Shared Systems," Journal of the Association for Computing Machinery, Vol. 15, No. 4, Oct. 1968, p. 557, equation (16).

[2] See Morse (1958), p. 10, or Avi Itzhak and Naor (1963) for discussions of equation (A.9).



The probability that Q_r will be greater than some value t, given that Y=y, is just the probability that X will be greater than y+t, given that it is already equal to y.

$$(A.10)\quad P(Q_r > t \mid Y=y) = P(X > t+y \mid X > y) = \frac{1 - F_X(t+y)}{1 - F_X(y)} \; ; \quad t,y \geq 0$$

By subtracting this result from unity one gets the conditional distribution function for Q_r given Y.

$$(A.11)\quad F_{Q_r|Y}(t) = P(Q_r \leq t \mid Y=y) = \frac{F_X(t+y) - F_X(y)}{1 - F_X(y)} \; ; \quad t,y \geq 0$$

$$(A.12)\quad P(t \leq Q_r \leq t+dt \mid Y=y) = \frac{F_X(t+dt+y) - F_X(t+y)}{1 - F_X(y)} \; ; \quad t,y \geq 0$$

Multiplying equation (A.12) by (A.9) leads to the following joint probability:

$$(A.13)\quad P(t \leq Q_r \leq t+dt \text{ and } y \leq Y \leq y+dy) = \frac{(F_X(t+dt+y) - F_X(t+y))\,dy}{E(X)} \; ; \quad t,y \geq 0$$

By integrating equation (A.13) over all values of Y, one gets the probability of the single event, $(t \le Q_r \le t+dt)$. Using the definition of $F_X(\cdot)$, one sees that the numerator of (A.13), integrated over all values of Y is just dt multiplied by the probability that X will be greater than t $(P(X > t) = 1 - F_X(t))$.

$$(A.14)\quad P(t \le Q_r \le t+dt) = \int_{y=0}^{\infty} P(t \le Q_r \le t+dt \text{ and } y \le Y \le y+dy)\cdot dy$$

$$= \frac{(1 - F_X(t))\cdot dt}{E(X)}, \quad t \ge 0$$

Equation (A.14) is the result presented earlier as equation (2.8). The first moment of Q_r was called ql_r in Chapter 2. The value of ql_r is easily obtained from (A.14) by using equation (A.3).

$$(A.15)\quad E(Q_r) = ql_r = \int_0^{\infty} \frac{t\cdot(1 - F_X(t))\cdot dt}{E(X)}$$

$$= E(X^2)/(2\cdot E(X))$$

The Laplace transform of Q_r is:

$$(A.16)\quad L_{Q_r}(s) = E(e^{-sQ_r}) = \int_{t=0}^{\infty} \frac{e^{-st}\cdot(1 - F_X(t))\cdot dt}{E(X)}$$

$$= \frac{1 - L_X(s)}{s\ E(X)}$$

Moments may be obtained from Laplace transforms of random variables by differentiation.

$$(A.17)\quad E(Q_r^k) = (-1)^k \cdot \left.\frac{d^k L_{Q_r}(s)}{ds^k}\right|_{s=0}$$

Using equation (A.17) one may show that the following result is valid for the k^{th} moment of Q_r.

$$(A.18) \quad E(Q_r^k) = \frac{E(X^{k+1})}{(k+1)\cdot E(X)} \quad , \quad k=1,2,\ldots$$

Equation (A.18) was presented previously as equation (2.9).

A4. DERIVATION OF POISSON SOURCE, EXPONENTIAL SERVICE, AND CONSTANT OVERHEAD, MODEL

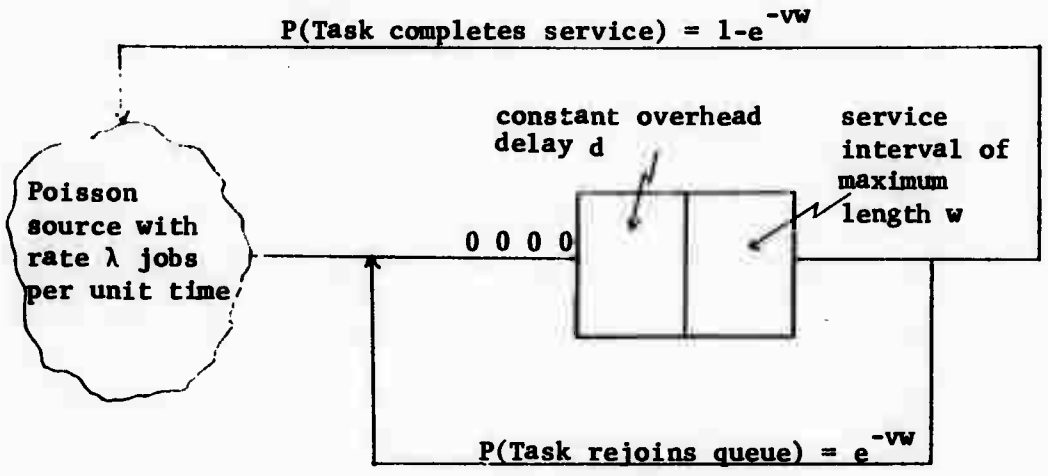
An assertion of Chapter 2 is that the methodology based on simple expected value arguments will handle more complex models than those previously published. The attempt by Rasch (1970) to use this technique in an exponential model with constant overhead was unsuccessful not because the method lacked power, but because Rasch failed to recognize important state dependent relationships. He neglected the fact that T_i, the wait in queue preceding service quantum i, is dependent on T_{i-1}.

Adiri and Avi-Itzhak (1969) solve the Poisson source, exponential service model with a constant overhead delay associated with every quantum. Figure A1 illustrates the structure of this model. The distribution function of the length of a quantum, Q, follows.

$$(A.19) \quad F_Q(t) = \begin{cases} 0 & t < d \\ 1-e^{-u(t-d)} & T \le t \le d+w \\ 1 & t < d+w \end{cases}$$

Every quantum is divided into two segments. The first interval, representing overhead delay, is a constant d, and the second is a random variable representing useful processing time. The maximum length of the second interval is a constant, w. The total processing request, V, exclusive of

overhead delay, is exponentially distributed with mean 1/v. If a request is not satisfied in a quantum, the job leaves the processor and rejoins the end of the queue. If a request is completed within the time limit w, the task leaves the system and a new quantum may start.



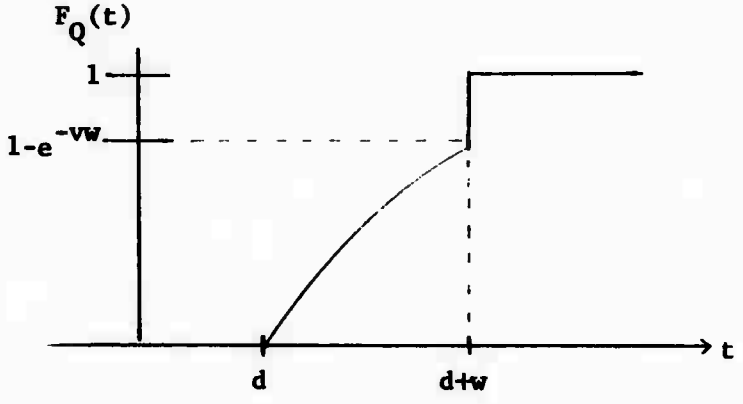


Figure A1
Structure of the Adiri/Avi-Itzhak Model

Adiri and Avi-Itzhak employ generating functions and Laplace transforms in non trivial ways in their derivation. The purpose of this section is to show that careful application of simpler, more easily understood,

techniques will also yield exact solutions to problems of this complexity. Transform techniques are more powerful than the methods of Chapter 2 since higher moments may be computed by differentiation. Transforms may also be numerically inverted to obtain the distribution function of a random variable. The price paid for this greater power is increased complexity which makes the results harder to understand and use.

The model of Figure A1 is another example of an M/G/1 queueing system. Like the early part of Section 2.3, the first part of the Adiri/Avi-Itzhak work is the calculation of total service request, expected number of tasks in the system, and the first two moments of the quantum interval. In the following discussion (AA.n) will denote equation n of the Adiri/Avi-Itzhak work. Minor symbol changes maintain notational compatability with Chapter 2.

Define: $\alpha = e^{-vw}$

(AA.12) $E(V) = 1/v + d/(1-\alpha)$

(AA.19) $\rho = 1-p_0 = \lambda \cdot E(V) < 1$

(AA.21) $E(M) = \rho + \lambda^2 E(V^2)/(2(1-\rho))$, where M is the random number of tasks in the queue and in the server

The expected wait in queue until the task enters the server for its first quantum interval, $E(T_1)$, given in equation (2.31), has the same derivation in this model as in the system of Section 2.3.4. Equation (2.31) is identical to (AA.49).

Adiri and Avi-Itzhak base their recursive equations on the random variable, K_i, the number of queued tasks <u>behind</u> the tagged job as it enters the server for the i^{th} quantum. $E(K_1)$ is the sum of three terms: (a) = the expected number of arrivals from the exponential input source during T_1;

(b) = the expected number of jobs that return to the queue that were in the queue in front of the tagged job when it arrived; and (c) = the expectation that the arriving job finds the server busy (Probability ρ) and that the job in service returns to the queue.

$$(A.20)\quad E(K_1) = (a) + (b) + (c)$$

$$(a) = \lambda \cdot E(T_1) = \lambda \cdot q1 \cdot (E(M)-\rho) + \lambda \cdot \rho \cdot q2/(2 \cdot q1)$$

$$(b) = e^{-v \cdot w} \cdot (E(M)-\rho) = \alpha \cdot (E(M)-\rho)$$

(c) requires careful treatment analogous to the derivation of $q1_r$, the expected remaining service time of the job in service when an arrival occurs. The probability that this task will return to the queue is not α as it is for the other jobs. The fact that this job has received a random amount of service when an arrival takes place alters the probability that it will return for additional quanta. Shemer (1967) neglected this fact as well as the fact that $q1_r \neq q1$.

Multiplying equation (A.11) by (A.9) gives one the probability of the joint events that (A) an arriving task finding the system busy enters y time units after the start of the service quantum in process and (B) waits less than t time units until that quantum is finished.

$$(A.21)\quad P(y \leq Y \leq y+dy \text{ and } Q_r \leq t) = \frac{\{F_Q(t+y) - F_Q(y)\}}{q_1}$$

Evaluating this expression for t = d+w-y and integrating over the allowable range of y values (0,d+w) gives the probability that a job in progress when a new arrival occurs finishes before the maximum quantum limit and therefore <u>does not</u> rejoin the queue. Call this event "D".

$$(A.22)\quad P(D) = \frac{1}{q1}\int_0^{d+w}\{F_Q(d+w) - F_Q(y)\}dy$$

$$= \frac{1}{q1}\int_0^{d+w}\{1-e^{-vw} - F_Q(y)\}dy$$

$$= \frac{q1 - (d+w)\cdot e^{-vw}}{q1}$$

The probability that the job does rejoin the queue is thus:

$$(A.23)\quad 1-P(D) = \frac{(d+w)\cdot\alpha}{q1}$$

Multiplying this result by ρ, the probability that the server is busy when a new job arrives, and substituting in equation (A.20) leads to the expected number of tasks behind a job as it enters the server for the first time.

$$(A.24)\quad E(K_1) = (E(M)-\rho)\cdot(\alpha+\lambda\cdot q1) + \rho\cdot\{\lambda\cdot q2 + 2\cdot\alpha\cdot(d+w)\}/(2\ q1)$$

To get this result in the exact form of Adiri and Avi-Itzhak one need only expand the quantity $\lambda q1$ and use eq. (AA.12) and eq. (AA.19).

$$(A.25)\quad \lambda q1 = \lambda\cdot(d+(1-\alpha)/v) = \lambda\cdot E(v)\cdot(1-\alpha) = \rho\cdot(1-\alpha)$$

Substituting $\rho\cdot(1-\alpha)$ for $\lambda\cdot q1$ in equation (A.24) leads to (AA.39). The authors note that their form of the result was derived from a generating function "after a rather lengthy and not painless process."[3]

The equations for $E(K_i)$ and $E(T_i)$ for i=2,3,4,... are easily derived, as in Chapter 2, once the initial values for i=1 are specified (equations (A.25) and (2.31)).

---

[3] Adiri and Avi-Itzhak (1969), p. 644.

APPENDIX B

LISTING OF TSMOD2

```
BEGIN
INTEGER LOOP;
FOR LOOP:=1 STEP 1 UNTIL 20 DO
SIMULA BEGIN
INTEGER INDEX,SEED,K,COUNT,STATE,STARTUP,MAXCUSTOMERS;
REAL SUMS,SUMSS,SUMC,SUMCC,SUMRS,SUMP,SUMPP;
REAL STATTIME,EXITPROB,STATEINTEGRAL,MARK,TEMP,OUT;
REAL MARK2,CPIDLE,CPREST;
REAL MARK3,IOIDLE,IOREST;
REAL ARRAY DATA(1:21,1:2),INT1(1:30),INT2(1:30);
INTEGER ARRAY TRANSITDIST(1:21),SERVICEDIST(1:21),STATEPROB(1:21);
INTEGER ARRAY INT3(1:30);
SET CPQUEUE,IOQUEUE;
ELEMENT PROCESSOR,IOSYS,MAIN;
BOOLEAN INITIALIZE;


PROCEDURE RESET;
BEGIN
     INITIALIZE:=FALSE;
     COUNT:=0;
     MARK:=MARK2:=STATTIME:=TIME;
     SUMS:=SUMSS:=SUMC:=SUMCC:=SUMRS:=SUMP:=SUMPP:=0.0;
     STATEINTEGRAL:=CPIDLE:=IOIDLE:=0.0;
     FOR K:=1 STEP 1 UNTIL 21 DO
          BEGIN
          TRANSITDIST(K):=SERVICEDIST(K):=STATEPROB(K):=0;
          DATA(K,1):=DATA(K,2):=0.0;
          END;
END OF PROCEDURE RESET;
```

```
ACTIVITY TASK ;
BEGIN
REAL ARRIVALTIME,SERVICETIME,CYCLE1;
BOOLEAN PASS1;
     PASS1:=TRUE;
     ARRIVALTIME:=TIME;
     HISTO(STATEPROB,INT3,STATE,(TIME-MARK)*100.0);
     ACCUM(STATEINTEGRAL,MARK,STATE,1);
     IF IDLE(PROCESSOR) THEN ACTIVATE PROCESSOR AFTER CURRENT;
     WAIT(CPQUEUE);
     TEMP:=TIME-ARRIVALTIME;
     SUMR:=SUMR+TEMP;
     SUMRR:=SUMRR+TEMP*TEMP;
     SUMS:=SUMS+SERVICETIME;
     SUMSS:=SUMSS+SERVICETIME*SERVICETIME;
     SUMC:=SUMC+CYCLE1;
     SUMCC:=SUMCC+CYCLE1*CYCLE1;
     SUMRS:=SUMRS+TEMP*SERVICETIME;
     HISTO(TRANSITDIST,INT1,TEMP,1);
     HISTO(SERVICEDIST,INT2,SERVICETIME,1);
     HISTO(STATEPROB,INT3,STATE,(TIME-MARK)*100.0);
     INDEX:=SERVICETIME*20.0+1.0;
     ACCUM(STATEINTEGRAL,MARK,STATE,-1);
     INDEX:=IF INDEX LEQ 21 THEN INDEX ELSE 21;
     DATA(INDEX,1):=DATA(INDEX,1)+TEMP;
     DATA(INDEX,2):=DATA(INDEX,2)+1.0;
     COUNT:=COUNT+1;
     IF (INITIALIZE AND COUNT EQL STARTUP) THEN RESET;
     IF COUNT EQL MAXCUSTOMERS THEN ACTIVATE MAIN;
END OF ACTIVITY TASK;


ACTIVITY GENERATOR;
BEGIN
 G1:HOLD(NEGEXP(1.0,SEED));
     ACTIVATE NEW TASK AFTER CURRENT;
     GO TO G1;
END OF ACTIVITY GENERATOR;
```

```
ACTIVITY COMPUTER;
BEGIN
REAL OVERHEAD,QUANTA;
 C1:EXTRACT FIRST(CPQUEUE) WHEN TASK DO
          BEGIN
       C2:OVERHEAD:=NORMAL(.05,.015,SEED);
          IF PASS1 THEN
               BEGIN
               CYCLE1:=TIME-ARRIVALTIME+OVERHEAD;
               PASS1:=FALSE;
               END;
          QUANTA:=NORMAL(.05,.015,SEED);
          SERVICETIME:=SERVICETIME+QUANTA;
          HOLD(OVERHEAD+QUANTA);
          INCLUDE(TASK,IOQUEUE);
          IF IDLE(IOSYS) THEN ACTIVATE IOSYS AFTER CURRENT;
          END
    OTHERWISE BEGIN
               ACCUM(CPIDLE,MARK2,CPREST,1.0);
               PASSIVATE;
               ACCUM(CPIDLE,MARK2,CPREST,-1.0);
          END;
    GO TO C1;
END OF ACTIVITY COMPUTER;


ACTIVITY IOPROCESSOR;
BEGIN
REAL IOSERVICE;
IO1: EXTRACT FIRST(IOQUEUE) WHEN TASK DO
          BEGIN
          IOSERVICE:=UNIFORM(0.0,.2,SEED);
         HOLD(IOSERVICE);
          TEMP:=UNIFORM(0.0,1.0,SEED);
          IF TEMP LEQ EXITPROB THEN ACTIVATE TASK AFTER CURRENT
               ELSE BEGIN
                    INCLUDE(TASK,CPQUEUE);
                    IF IDLE(PROCESSOR) THEN
                         ACTIVATE PROCESSOR AFTER CURRENT;
                    END;
          END
    OTHERWISE BEGIN
          ACCUM(IOIDLE,MARK3,IOREST,1.0);
          PASSIVATE;
          ACCUM(IOIDLE,MARK3,IOREST,-1.0);
          END;
GO TO IO1;
END OF ACTIVITY IOPROCESSOR;
```

```
END OF ACTIVITY IOPROCESSOR;


READ (SEED,STARTUP,MAXCUSTOMERS);
WRITE('  SEED:=',SEED,'  STARTUP:=',STARTUP,'MAXCUSTOMERS:=',MAXCUSTOMERS);
EXITPROB:=.125;
INITIALIZE:=TRUE;
FOR K:=1 STEP 1 UNTIL 20 DO
    BEGIN
    INT1(K):=1.5*K;
    INT2(K):=.1*K;
    INT3(K):=K-1;
    END;
PROCESSOR := NEW COMPUTER;
IOSYS:=NEW IOPROCESSOR;
MAIN:=CURRENT;
ACTIVATE NEW GENERATOR AFTER CURRENT;
PASSIVATE;
ACCUM(STATEINTEGRAL,MARK,STATE,0);
ACCUM(CPIDLE,MARK2,CPREST,0.0);
ACCUM(IOIDLE,MARK3,IOREST,0.0);
WRITE(' TIME AT RUN COMPLETION IS',TIME,' NUMBER OF TASKS PROCESSED IS',
     COUNT);
WRITE(' CURRENT STATE IS',STATE,'AVERAGE STATE IS',
     STATEINTEGRAL/(TIME-STATTIME));
TEMP:=SUMS/COUNT;
OUT:=SQRT(SUMSS/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF SERVICE TIMES ARE',TEMP,OUT);
TEMP:=(COUNT*SUMRS-SUMR*SUMS)/(COUNT*SUMSS-SUMS*SUMS);
OUT:=(SUMR-TEMP*SUMS)/COUNT;
WRITE(' REGRESSION ESTIMATE OF SLOPE AND INTERCEPT ARE',TEMP,OUT);
TEMP:=SUMC/COUNT;
OUT:=SQRT(SUMCC/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF FIRST WAITING TIMES ARE',TEMP,OUT);
TEMP:=SUMR/COUNT;
OUT:=SQRT(SUMRR/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF RESPONSE TIMES ARE',TEMP,OUT);
WRITE(' PROBABILITY PROCESSOR IS IDLE IS',CPIDLE/(TIME-STATTIME));
WRITE(' PROBABILITY IOSYSTEM IS IDLE IS',IOIDLE/(TIME-STATTIME));
WRITE(' THE TABLE DISPLAYS RESPONSE AS A FUNCTION OF SERVICE');
WRITE(' INTERVAL','AVERAGE R','NUMBER OF POINTS');
FOR K:=1 STEP 1 UNTIL 21 DO
    BEGIN
    TEMP:=IF DATA(K,2) GEQ 1.0 THEN DATA(K,1)/DATA(K,2) ELSE 0.0;
    OUT:=K/20;
    WRITE(OUT,TEMP,DATA(K,2));
    END;
END OF SIMULA BLOCK;
END OF PROGRAM;
```

APPENDIX C
LISTING OF TSS/360 MODEL

```
BEGI
INTEGER STARTCLOCK,SIMTIME,THINK,WSMEAN;
EXTERNAL PROCEDURE LISTQ,LPRINT,MAKESCHEDULETABLE;
REAL OVERHEAD,VMTMEAN;
BOOLEAN WAIT,IODFLT,PRWAIT,IDLING;
INTEGER MAXUSERS,
        SCHEDTABENTRIES,
        MAXTASKSONDISPLIST,LOSPAGES,CHANNELS;
   SCHEDTABENTRIES = 29;
BEGI
   INTEGER ARRAY SCHEDTAB(1:SCHEDTABENTRIES,1:9);
   INTEGER NUMOFTASKS, PAGESUSED,QUANTA,IOLEVEL,
           PRIORITY, TIMESLICE,MAXCR,
           MAXPGRD, DTR, TSEND, MPRE , PAGEDELAY;
   INTEGER SEED1,SEED2,SEED3,TRACELEVEL,SEED4,SEED5,SEED6,SEED7;
   LIST DATA(CHANNELS,LOSPAGES,PAGEDELAY,MAXTASKSONDISPLIST,MAXUSERS,
             TRACELEVEL,SIMTIME,OVERHEAD,THINK,VMTMEAN,WSMEAN);
   FORMAT DATAPRINT(I3,' CHANNELS',I4,' LOS PAGES',I5,' MSEC PAGE DELAY'
                    ,I3,' DISP TASKS',A1,1,I3,' USERS',I4,' LEVEL TRACE',
                     I3,' MINS SIMULATED TIME',D9,3,' OVERHEAD FACTOR',
                    A1,1,I3,' SECS THINK TIME',D8,1,' MSEC COMPUTE TIME',
                    I5,' PAGE WORK SET',A1,1);
   LOCAL LABEL NODATALEFT;
   PRIORITY=1;
   TIMESLICE=2;
   QUANTA=3;
   MAXCR=4;
   MAXPGRD=5;
   DTR=6;
   TSEND=7;
   MPRE =8;
   IOLEVEL=9;
   MAKESCHEDULETABLE(SCHEDTABENTRIES,SCHEDTAB);
GETDATA;
READ(DATA, NODATALEFT);    WRITE(DATA,DATAPRINT);
BEGIN
INTEGER ARRAY PAGERQNUM,PAGERQHOWMANY(1:MAXUSERS*2);
INTEGER PAGEENTRY,SEED;
FOR SEED=2*03 DO
SIMULA BEGIN
REAL STARTIDLETIME;
ARRAY TRIV,AVG,HARD,STARTPAGE,ALL,IDLETIME(-2:3*1)  ;
ARRAY ACTUALQUANTLENGTH(-2:1*1);
ARRAY FAULTS(-2:51);
FORMAT GABR('USR',D9,2,'  USER',2I3,'  RESP',D6,1,'  COMP',D5,3,
            '  WRKS',I3,'  UPRG',I3,'  FAULTS',I3,'  DISP',I3,
            '  ELIG',I3,'  TASKS',I3,'  USRPG',I3,'  TOTPG',I4,A1);
   INTEGER PROCEDURE FAIRSHARE;
      FAIRSHARE=ENTIER(LOSPAGES/NUMOFTASKS);
      ELEMENT ELINTERNALSCHEDULER,
              ELTIMER,
              ELQUEUESCANNER,
              ACTIVETASK;
      ELEMENT ARRAY ELUSER,TSI(1:MAXUSERS),ELPAGEPROCESSOR(1:CHANNELS);
      SET DISPIO, DISPEX,
          ELIGIBLELIST,
          INACTIVELIST;
ACTIVITY TASK( USERNUM );
   INTEGER USERNUM;
   BEGIN
   FORMAT PGOUT('PTW',D9,2,'  TASK',I3,I7,' PAGESOUT AT TWAIT',A1);
```

```
    BOOLEAN PAGEWAIT, IOBOUND;
    REAL VMTIME,SST,TIMECOUNT,STARTTIME,TIMELEFT,STARTPAGEWAIT;
    INTEGER QUANTAUSED,WORKINGSET, PAGEFAULTS;
    INTEGER STE, NEWPAGESREQ, PAGECOUNT, PAGESOUT, NEWPAGE,PAGESTOGET;
       BEGIN
       REAL VMTIMEPIECE;
      VERYTOPOFTASK;
       PAGEFAULTS=QUANTAUSED=0;
       TIMELEFT=VMTIME;
       PAGESTOGET = NEWPAGESREQ;
       TOPOFTASK;
       VMTIMEPIECE=TIMELEFT/(PAGESTOGET+1);
       FOR NEWPAGE=(0,1,PAGESTOGET) DO
          BEGIN
             IF TIMECOUNT+VMTIMEPIECE LSS SCHEDTAB(STE,TIMESLICE) THEN
                BEGIN
                CANCEL(ELTIMER);
                HOLD(VMTIMEPIECE);
                LISTO(ACTUALQUANTLENGTH,0,100,0,5,VMTIMEPIECE);
                TIMECOUNT=TIMECOUNT+VMTIMEPIECE;
                END
                ELSE BEGIN
                COMMENT  TIMER INTERRUPT COMES HERE   ;
                LISTO(ACTUALQUANTLENGTH,0,100,0,5,
                        SCHEDTAB(STE,TIMESLICE)-TIMECOUNT);
                PASSIVATE;
                GO TO TOPOFTASK;
                END;
          IF NEWPAGE EQL PAGESTOGET THEN GO TO TWAIT;
          PAGEENTRY=PAGEENTRY+1;
          PAGERQHOWMANY(PAGEENTRY)=1;
          PAGERQWHO(PAGEENTRY)=USERNUM;
          PAGEFAULTS=PAGEFAULTS+1;
          PAGEWAIT=TRUE;
          STARTPAGEWAIT=TIME;
          ACTIVATE ELQUEUESCANNER DELAY PAGEDELAY;
          PASSIVATE
          END;
       TWAIT;
       COMMENT  ISSUE TWAIT ;
       PAGESOUT = PAGECOUNT - FAIRSHARE;
       IF PAGESOUT GTR 0 THEN BEGIN
          STARTPAGEWAIT=0;
          PAGEENTRY=PAGEENTRY+1;
          PAGERQWHO(PAGEENTRY)=USERNUM;
          PAGERQHOWMANY(PAGEENTRY)=-PAGESOUT;
       IF TRACELEVEL GTR 2 THEN WRITE(TIME/1000,USERNUM,PAGESOUT,PGOUT);
       END OF PAGINGOUT;
       ACTIVATE ELUSER(USERNUM) AFTER CURRENT;
       ACTIVATE ELQUEUESCANNER AFTER CURRENT;
       TRANSFER(TSI(USERNUM), INACTIVELIST);
       TIMECOUNT=0;
       PASSIVATE;
       GO TO VERYTOPOFTASK
       END
    END;
PROCEDURE FILEINELIGLIST( LTSI );
    ELEMENT LTSI;
    INSPECT LTSI WHEN TASK DO
       BEGIN ELEMENT TSIINLIST;
       INTEGER LTSIPRI; REAL LTSISST;LOCAL LABEL OUT;
```

```
      LTSIPRI=SCHEDTAB(STE,PRIORITY);
      LTSISST=SST;
      TSIINLIST=HEAD(ELIGIBLELIST);
      FOR TSIINLIST=SUC(TSIINLIST) WHILE EXIST(TSIINLIST) DO
         INSPECT TSIINLIST WHEN TASK DO
            IF LTSIPRI LSS SCHEDTAB(STE,PRIORITY) THEN
               BEGIN PRECEDE(TSIINLIST,LTSI);
               GO TO OUT END
               ELSE
               IF LTSIPRI EQL SCHEDTAB(STE,PRIORITY) AND
                  LTSISST LSS SST THEN
                  BEGIN PRECEDE(TSIINLIST,LTSI);
                  GO TO OUT END;
      TRANSFER(LTSI,ELIGIBLELIST);
      OUT ! END OF FILEINELIGLIST;
ACTIVITY TIMER;
   BEGIN
   BOOLEAN FORCED;
   FORMAT TYME('TMR',D9,2,'  TASK',I3,X3,S2,X2,S2,X3,'PRIORITY= ',I3,
               X2,'SST= ',D6,1,X2,'STE= ',I3,X2,S15,A1),
          PAGOUT('PSE',D9,1,'  TASK',I3,I7,' PAGESOUT AT TSEND',I6,
               ' PAGES NEXT TIME ',A1);
   TOPOFTIMER;
   FORCED=FALSE;
   INSPECT ACTIVETASK WHEN TASK DO
      BEGIN BOOLEAN OLDIO;  INTEGER PAGESOUT;  LOCAL LABEL L;
      OLDIO=IOBOUND;
      TIMELEFT=TIMELEFT-SCHEDTAB(STE,TIMESLICE);
      QUANTAUSED=QUANTAUSED + 1;
      IF QUANTAUSED GEQ SCHEDTAB(STE,QUANTA) OR
         PAGECOUNT GTR SCHEDTAB(STE,MAXCR) OR
         NEWPAGE GEQ SCHEDTAB(STE,MAXPGRD) THEN BEGIN
      STE=SCHEDTAB(STE,IF PAGECOUNT GTR SCHEDTAB(STE,MAXCR)
         THEN MPRE ELSE TSEND);
      SST=IF SCHEDTAB(STE,DTR) EQL 0 THEN 0 ELSE
         TIME+SCHEDTAB(STE,DTR)+
            IF SST LSS 0 THEN SST ELSE 0;
      CANCEL(ACTIVETASK);
      FILEINELIGLIST(ACTIVETASK);
      PAGESOUT=PAGECOUNT-FAIRSHARE;
      FORCED=TRUE;
      IF PAGESOUT GTR 0 THEN
         BEGIN
         STARTPAGEWAIT=0;
         PAGEENTRY=PAGEENTRY+1;
         PAGEROWHO(PAGEENTRY)=USERNUM;
         PAGEROWHOWANY(PAGEENTRY)=PAGESOUT;
         PAGESTOGET=MAX(0,PAGESTOGET-NEWPAGE+
                           PAGESOUT*UNIFORM(.2,.7,SEED5));
        IF TRACELEVEL GTR 2 THEN
            WRITE(TIME/1000,USERNUM,PAGESOUT,PAGESTOGET,PAGOUT);
         GO TO L
         END OF PAGING OUT;
      END OF TIMESLICEEND;
      PAGESTOGET=PAGESTOGET-NEWPAGE;
      L  ;
      IOBOUND=IF NEWPAGE GEQ SCHEDTAB(STE,IOLEVEL)
         THEN TRUE ELSE FALSE ;
      TIMECOUNT=0;
IF TRACELEVEL GTR 1 THEN
WRITE(TYME,TIME/1000,USERNUM,IF OLDIO THEN 'IO' ELSE 'EX',
```

```
IF IOBOUND THEN 'IO' ELSE 'EX',SCHEDTAB(STE,PRIORITY),SST/1000,STE,
IF FORCED THEN 'TSEND FORCED' ELSE '  '));
      END;
   ACTIVATE ELQUEUESCANNER AFTER CURRENT;
   PASSIVATE;
   GO TO TOPOFTIMER
   END;
ACTIVITY USER(ID,PRTY,CONVERSE,THINKSECONDS,COMPUTEMEAN,
              TRANSACTIONS,WORKSETMEAN);
  INTEGER ID,PRTY,THINKSECONDS,TRANSACTIONS,WORKSETMEAN;
   BOOLEAN CONVERSE;
   REAL COMPUTEMEAN;
   BEGIN INTEGER  I ;
   REAL TEMP;
   TSI(ID)=NEW TASK(ID);
   NUMOFTASKS = NUMOFTASKS+1;
   FOR I=(1,1,TRANSACTIONS) DO
      BEGIN
      INSPECT TSI(ID) WHEN TASK DO
         BEGIN
         IOBOUND=IODFLT;
         STARTTIME=TIME;
         VMTIME=NEGEXP(1/COMPUTEMEAN,SEED1);
         WORKINGSET=RANDINT(0,2*WORKSETMEAN,SEED2);
         NEWPAGESREQ=MAX(0,WORKINGSET-RANDINT(0,PAGECOUNT,SEED5));
         STE=PRTY*IF CONVERSE THEN 0 ELSE 10;
         IF I EQL 1 THEN SST=0 ELSE
         SST=SCHEDTAB(STE,DTR)*TIME+
            IF SST LSS 0 THEN SST ELSE 0
                                                                  ;
      FILEINELIGLIST(TSI(ID));
      IF WAIT THEN BEGIN WAIT=FALSE;
            ACTIVATE ELQUEUESCANNER  AFTER CURRENT END;
      PASSIVATE  ;
IF CONVERSE THEN BEGIN
TEMP=(TIME-STARTTIME)/1000;
LISTO(ALL,0,300,2,0,TEMP);
LISTO(FAULTS,0,50,1,0,PAGEFAULTS+,01);
IF VMTIME LSS 35 AND NEWPAGESREQ LEQ 2 THEN
LISTO(TRIV,0,300,.25,TEMP) ELSE
IF VMTIME GTR 250 OR  NEWPAGESREQ GTR 10 THEN
LISTO(HARD,0,300,2,0,TEMP) ELSE LISTO(AVG,0,300,1,0,TEMP);
END OF COLLECTING DATA;
IF TRACELEVEL GTR 0 THEN
WRITE(GARR,TIME/1000,USERNUM,I,(TIME-STARTTIME)/1000,VMTIME/1000,
      WORKINGSET,NEWPAGESREQ,PAGEFAULTS,CARDINAL(DISPID)+
                                        CARDINAL(DISPEX),
      CARDINAL(ELIGIBLELIST),NUMOFTASKS,PAGECOUNT,PAGESUSED);
      HOLD( THINKSECONDS * 1000 );
END OF INSPECTION;
      END OF TRANSACTION LOOP;
   REMOVE(TSI(ID));
   TERMINATE(TSI(ID));
   NUMOFTASKS=NUMOFTASKS-1;
   TERMINATE(CURRENT);
   END;
ACTIVITY PAGEPROCESSOR(J);
   INTEGER J;
   BEGIN
   INTEGER USERNUM,NUMPAGES,I;
   TOPOFPAGEPROCESSOR;
```

```
      NUMPAGES=PAGEROWHOWMANY(1);
      USERNUM=PAGEROWHO(1);
      PAGEENTRY=PAGEENTRY-1;
      FOR I=(1,1,PAGEENTRY) DO BEGIN
         PAGEROWHO(I)=PAGEROWHO(I+1);
         PAGEROWHOWMANY(I)=PAGEROWHOWMANY(I+1); END;
            BEGIN  INTEGER I, PAGINGOPERATIONS;
            REAL GETREQUESTTIME;
            GETREQUESTTIME=TIME;
            PAGINGOPERATIONS=ABS(NUMPAGES)//8+1 ;
            FOR I=(1,1,PAGINGOPERATIONS) DO
               HOLD(UNIFORM(25,0,125,0,SEED3));
            PAGESUSED = PAGESUSED + NUMPAGES;
            INSPECT TSI(USERNUM) WHEN TASK DO
               BEGIN
               PAGEWAIT=FALSE;
               PAGECOUNT=PAGECOUNT+NUMPAGES  ;
               IF STARTPAGEWAIT GTR 0 THEN
                  LISTO(STARTPAGE,0,300,,2,
                                       (GETREQUESTTIME-STARTPAGEWAIT)/1000);
               END  ;
         IF PPWAIT THEN
            BEGIN
            PPWAIT=FALSE;
            ACTIVATE ELQUEUESCANNER AFTER CURRENT;
            IF IDLING THEN
               BEGIN
               IDLING=FALSE;
               LISTO(IDLETIME,0,300,,1,(TIME-STARTIDLETIME)/1000)
               END
            END;
            END;
      PASSIVATE;
      GO TO TOPOFPAGEPROCESSOR
      END;
ACTIVITY QUEUESCANNER;
      BEGIN INTEGER I;
      REAL LASTTIMETHRU;
      TOPOFQUEUESCANNER;
      FOR I=(1,1,CHANNELS) DO IF PAGEENTRY GTR 0 THEN
            ACTIVATE ELPAGEPROCESSOR(I);
      ACTIVATE ELINTERNALSCHEDULER DELAY (TIME-LASTTIMETHRU)*OVERHEAD;
      PASSIVATE;
      LASTTIMETHRU=TIME;
      GOTO TOPOFQUEUESCANNER
      END OF Q-SCANNER;
ACTIVITY INTERNALSCHEDULER;
      BEGIN
      ELEMENT LTSI;
      INTEGER PRI;
      BOOLEAN BEHINDONLY;
      BOOLEAN PROCEDURE ENTRANCECRITERIA;
         ENTRANCECRITERIA=
            IF CARDINAL(DISPIO)+CARDINAL(DISPEX) LSS MAXTASKSONDISPLIST
               THEN TRUE ELSE FALSE;
      PROCEDURE MOVETODISPLIST( LTSI );
         ELEMENT LTSI;
         INSPECT LTSI WHEN TASK DO
            BEGIN
            LOCAL LABEL FOUNDPLACE;
            SST=IF SST LSS 0 THEN 0 ELSE SST-TIME;
```

```
      IF IOBOUND THEN
         BEGIN
         INTEGER LTSIPRI;
         ELEMENT INIO;
         LTSIPRI=SCHEDTAB(STE,PRIORITY);
         FOR INIO=LAST(DISPIO),PRED(INIO) WHILE EXIST(INIO) DO
            INSPECT INIO WHEN TASK DO
               IF SCHEDTAB(STE,PRIORITY) LEQ LTSIPRI THEN
                  BEGIN  FOLLOW(INIO,LTSI);   GO FOUNDPLACE  END;
         PRECEDE(HEAD(DISPIO),LTSI)
         END IOBOUND CASE
         ELSE TRANSFER(LTSI,DISPEX);
      FOUNDPLACE;
   END OF MOVE TO DISP LIST;
LOCAL LABEL SORTDISPATCH;
TOPOFINTERNALSCHEDULER;
BEHINDONLY=TRUE;
TOPOFSEARCH;
IF EMPTY(ELIGIBLELIST) THEN GO TO SORTDISPATCH;
LTSI=FIRST(ELIGIBLELIST);
FOR LTSI=SUC(LTSI) WHILE EXIST(PRED(LTSI)) DO
   INSPECT PRED(LTSI) WHEN TASK DO
      IF NOT PAGEWAIT THEN
         BEGIN
         IF BEHINDONLY AND SST GEQ TIME
            THEN GO TO TSINOGOOD
            ELSE IF ENTRANCECRITERIA THEN
               BEGIN
               MOVETODISPLIST(TASK);
               GO TO TOPOFINTERNALSCHEDULER
               END
               ELSE GO TO SORTDISPATCH;
         TSINOGOOD;
         END;
IF NOT BEHINDONLY THEN GO TO SORTDISPATCH;
BEHINDONLY=FALSE;
GO TO TOPOFSEARCH;
SORTDISPATCH;
   IF NOT EMPTY(DISPEX) THEN TRANSFER(FIRST(DISPEX),DISPEX);
BEGIN
COMMENT
        THIS  IS THE DISPATCHER PART OF THE SCHEDULER
                                                              ;
INTEGER I;
BOOLEAN BUSYPAGERS;
ELEMENT DTSI;
FOR DTSI=FIRST(DISPIO),SUC(DTSI) WHILE EXIST(DTSI),
         FIRST(DISPEX),SUC(DTSI) WHILE EXIST(DTSI) DO
   INSPECT DTSI WHEN TASK DO
      IF NOT PAGEWAIT THEN
         BEGIN
         ACTIVETASK = DTSI;
         ACTIVATE ACTIVETASK AFTER CURRENT;
         ACTIVATE ELTIMER DELAY SCHEDTAB(STE,TIMESLICE)*TIMECOUNT;
         PASSIVATE   ;
         GO TO TOPOFINTERNALSCHEDULER;
         END;
BUSYPAGERS=FALSE;
FOR I=(1,1,CHANNELS) DO IF NOT IDLE(ELPAGEPROCESSOR(I)) THEN
   BUSYPAGERS=TRUE;
IF BUSYPAGERS THEN
```

```
      BEGIN
      PPWAIT=TRUE;
      IF NOT EMPTY(ELIGIBLELIST) THEN
         BEGIN
         IDLING=TRUE;
         STARTIDLETIME=TIME
         END
      END
    ELSE
      IF PAGEENTRY LEQ 0 THEN WAIT=TRUE ELSE
         ACTIVATE ELQUEUESCANNER AFTER CURRENT;
   PASSIVATE;
   GO TO TOPOFINTERNALSCHEDULER
   END OF DISPATCHER PART;
  END OF INTERNALSCHEDULER;
COMMENT    CONTROLLING CODE COMES NEXT;
   BEGIN
   INTEGER I;
   STARTCLOCK=CLOCK;
   PAGEENTRY=0;
   IDLING=FALSE;
   PPWAIT = FALSE ;
   IODFLT = TRUE;
   WAIT=TRUE;
   PAGESUSED = 0;
   SEED1=SEED;  SEED2=SEED1*17;  SEED3=SEED1*71;
   SEED4=SEED*23;   SEED5=SEED*31;   SEED6=SEED*57;  SEED7=SEED*51;
   NUMOFTASKS = 3;
   ELQUEUESCANNER = NEW QUEUESCANNER;
   ELTIMER = NEW TIMER  ;
   ELINTERNALSCHEDULER = NEW INTERNALSCHEDULER;
   FOR I=(1,1,CHANNELS) DO ELPAGEPROCESSOR(I)=NEW PAGEPROCESSOR(I);
   FOR I=(1,1,MAXUSERS) DO
      BEGIN
      ELUSER(I)=NEW USER(I,5,TRUE,THINK,VMTMEAN,50,WSMEAN);
      ACTIVATE ELUSER(I) AT I*3000
      END;
   END;
   HOLD( SIMTIME * 60000 );
WRITE('*********   REAL TIME IN SECS',CLOCK-STARTCLOCK,' ******');
LPRINT(ALL,0,300,2,0,1,0,'ALL');
LPRINT(TRIV,0,300,,25,1,0,'TRIVIAL');
LPRINT(AVG,0,300,1,0,,97,'AVERAGE');
LPRINT(HARD,0,300,2,0,,97,'HARD');
LPRINT(STARTPAGE,0,300,,2,1,0,'TIME TO START PAGE FETCH');
LPRINT(IDLETIME,0,300,,1,1,0,'IDLE TIME WHILE ELIGIBLE TASKS');
LPRINT(FAULTS,0,50,1,0,1,0,'PAGE FAULTS');
LPRINT(ACTUALQUANTLENGTH,0,100,0,5,,99,'ACTUAL COMPUTE SLICES');
END OF SIMULA BLOCK
END OF MAXUSERS LOOP;
GO GETDATA;
!DATALEFT;
END OF SCHEDTAB DECL
END OF EVERYTHING
**********                    **********
```

APPENDIX D

LISTING OF SCRIPT PROGRAMS

Figure 4.1 is a block diagram of the User Script used for the response time experiments reported in Sections 4.2.1 and 4.3. Figure 4.3 is a listing of TEST1, the first program in the script. The following two listings are of TEST2 and TEST3. The terminal output of an actual user following the script is also included. The arrows on this latter listing indicate the places where users must insert input data to the script system. This example output was used as part of the training of the users who participated in the experiment.

```
      DIMENSION IB(8),IE(8)
    2 I=0
      PRINT 899
  899 FORMAT(/'PLEASE ENTER N IN FORM J*10**K')
      READ 900,N,NN
  900 FORMAT(2I1)
      N=N*10**NN
      IF (N.EQ.0)  GO TO 10
      PRINT 901,N
  901 FORMAT ('TEST2: CYCLES=',I7)
      CALL CLOCK(IB)
      DO 1 J=1,N
    1 I=I+1
      CALL CLOCK(IE)
      TRAN=36000*(IE(1)-IB(1))+3600*(IE(2)-IB(2))+600*(IE(3)-IB(3))
      TRAN=TRAN+60*(IE(4)-IB(4))+10*(IE(5)-IB(5))+(IE(6)-IB(6))
      TRAN=TRAN+.1*(IE(7)-IB(7))+.01*(IE(8)-IB(8))
      PRINT 903,IB
  903 FORMAT('START TIME= ',2I1,':',2I1,':',2I1,'.',2I1)
      PRINT 903,IE
  904 FORMAT('END TIME=',2I1,':',2I1,':',2I1,'.',2I1)
      PRINT 905,TRAN
  905 FORMAT('RESPONSE TIME=',F8.2)
      GO TO 2
   10 PRINT 906
  906 FORMAT('TEST2 NOW COMPLETE.  YOU ARE IN COMMAND MODE')
      STOP
      END
```

```
      REAL M1,M2,M3,M
      DIMENSION IB(8),IE(8),C(20000)
      DIMENSION M1(30,30),M2)30,30),M3)30,30)
  100 PRINT 899
  899 FORMAT(/'TEST3:  PLEASE ENTER N IN FORMAT 12')
      READ 900,N
  900 FORMAT(I2)
      IF (N.EQ.0)  GO TO 10
      PRINT 901,N
  901 FORMAT('TEST3: ITERATIONS=',I3)
      CALL CLOCK(IB)
      DO 6 L=1,N
      DO 50 I=1,20000
   50 C(I)=I
      DO 2 I=1,30
      DO 1 J=1,30
      M1(I,J)=2
      M2(I,J)=3
    1 CONTINUE
    2 CONTINUE
      DO 5 I=1,30
      DO 4 J=1,30
      M=0
      DO 3 K=1,30
    3 M=M+M2(I,K)*M1(K,J)
      M3(I,J)=M
    4 CONTINUE
    5 CONTINUE
    6 CONTINUE
      CALL CLOCK(IE)
      TRAN=36000*(IE(1)-IB(1))+3600*(IE(2)-IB(2))+600*(IE(3)-IB(3))
      TRAN=TRAN+60*(IE(4)-IB(4))+10*(IE(5)-IB(5))+(IE(6)-IB(6))
      TRAN=TRAN+.1*(IE(7)-IB(7))+.01*(IE(8)-IB(8))
      PRINT 903,IB
  903 FORMAT('START TIME=',2I1,':',2I1,':',2I1,'.',2I1)
      PRINT 904,IE
  904 FORMAT('END TIME=',2I1,':',2I1,':',2I1'.',2I1)
      PRINT 905,TRAN
  905 FORMAT('RESPONSE TIME=',F8.2)
      GO TO 100
   10 PRINT 906
  906 FORMAT('TEST3 NOW COMPLETE.  YOU ARE IN COMMAND MODE')
      STOP
      END
```

```
B001 5.1 TSS AT CMU  TASKID=0022 04/07/70 19:02 3900 SDA=0064
→USER00
18:56 07 APR 70-BENCHMARK TESTS ARE TO BE CONDUCTED AT 1900 - 2100 ..


→SHARE PUBPRO,S132JM23,PUBPRO
→CDS PUBPRO,USERLIB(SYSPRO)
CZAFV460 ENTER DSORG VIP OR VSP. DEFAULT: MEMBER GIVEN OLD DSORG.

→ABEND
≠~'( 'ɔ)ɔ≠ɔ' ≥v 16↑-+-+-'
⊢ɔc ≠~'( )÷~>ɔ' ↑⊢. ≠~'(↓'! 0023

→KU
→INITIAL
CZUHW001 AT 19:04:37: 6 USERS, 0 TTYVS, 6 2741VS, 0 1050VS
BF03 SOURCE.TESTA COPIED AS SOURCE.TEST1.
BF03 SOURCE.TESTB COPIED AS SOURCE.TEST2.
BF03 SOURCE.TESTC COPIED AS SOURCE.TEST3.
BF03 SOURCE.FFCLOCK COPIED AS SOURCE.FCLOCK.
CZU4W001 AT 19:04:53: 6 USERS, 0 TTYVS, 6 2741VS, 0 1050VS
A053 MODIFICATIONS? ENTER Y OR N.
→N
B016 LP FOUND NO ERRORS.
0000250 E *** ILLEGAL EXPRESSION. OPERAND NOT FOUND WHERE REQUIRED
0000250  X=Y(
□


LINE NOISE. REENTER


0000251 E *** ERROR IN RELATION OR LOGICAL OPERATOR OR CONSTANT
0000251  Z=X.Y
□
A053 MODIFICATIONS? ENTER Y OR N.
→Y
→□D,250,251
→□R,100,2100
0000100 DIMENSION IB(8),IE(8)
0000200 DIMENSION M1(30,30),M2(30,30),M3(30,30)
0000300  100 PRINT 899
0000400 899 FORMAT(/V  TEST1: PLEASE ENTER N IN FORMAT I2V)
0000500 READ 900,N
0000525 900 FORMAT(I2)
0000550 IF (N.EQ.0)  GO TO 10
0000600 PRINT 901,N
0000700 901 FORMAT(V  TEST1: ITERATION=V,I3)
0000800 CALL CLOCK(IB)
0000900 DO 6 L=1,N
0000910 DO 2 I=1,30
0000920 DO 1 J=1,30
0000930 M1(I,J)=2
0000940 M2(I,J)=3
0000950 1 CONTINUE
0000960 2 CONTINUE
0000970 DO 5 I=1,30
0000980 DO 4 J=1,30
0000990 M=0
```

```
0001000 DO 3 K=1,30
0001010 3 M=M+M2(I,K)*M1(K,J)
0001020 M3(I,J)=M
0001030 4 CONTINUE
0001040 5 CONTINUE
0001050 6 CONTINUE
0001100 CALL CLOCK(IE)
0001200 TRAN=36000*(IE(1)-IB(1))+3600*(IE(2)-IB(2))+600*(IE(3)-IB(3))
0001300 TRAN=TRAN+60*(IE(4)-IB(4))+10*(IE(5)-IB(5))+(IE(6)-IB(6))
0001400 TRAN=TRAN+.1*(IE(7)-IB(7))+.01*(IE(8)-IB(8))
0001700 PRINT 903,IB
0001701 903 FORMAT(▽  START TIME= ▽,211,▽:▽,211,▽:▽,211,▽.▽,211)
0001702 PRINT 904,IE
0001703 904 FORMAT(▽  END   TIME= ▽,211,▽:▽,211,▽:▽,211,▽.▽,211)
0001704 PRINT 905,TRAN
0001800 905 FORMAT(▽  RESPONSE TIME= ▽,F8.2)
0001900 WRITE(1,800) N,IB,IE,TRAN
0001950800 FORMAT(2H 1,I8,8I1,8I1,F8.2)
0002000 GO TO 100
0002050 10 PRINT 906
0002051 906 FORMAT(▽  TEST1 NOW COMPLETE. YOU ARE IN COMMAND MODE▽)
0002052 STOP
0002100 END
]
A053 MODIFICATIONS? ENTER Y OR N.
→N
B016 LP FOUND NO ERRORS.
→TEST1

 TEST1: PLEASE ENTER N IN FORMAT I2
→01
 TEST1: ITERATION=  1
 START TIME= 19:08:49.86
 END   TIME= 19:08:52.36
 RESPONSE TIME=     2.50

 TEST1: PLEASE ENTER N IN FORMAT I2
→10
 TEST1: ITERATION= 10
 START TIME= 19:09:05.15
 END   TIME= 19:09:34.35
 RESPONSE TIME=    29.20

 TEST1: PLEASE ENTER N IN FORMAT I2
→00
 TEST1 NOW COMPLETE. YOU ARE IN COMMAND MODE
 CHCIW STOP
→FTN TEST2,Y
A053 MODIFICATIONS? ENTER Y OR N.
→N
B016 LP FOUND NO ERRORS.
→TEST2

TEST2: PLEASE ENTER N IN FORM J*10**K
→11
 TEST2: CYCLES=     10
 START TIME= 19:10:49.68
 END   TIME= 19:10:49.70
 RESPONSE TIME=     0.02

TEST2: PLEASE ENTER N IN FORM J*10**K
```

```
→25
 TEST2: CYCLES= 200000
 START TIME= 19:11:14.36
 END   TIME= 19:12:08.27
 RESPONSE TIME=    53.91

TEST2: PLEASE ENTER N IN FORM J*10**K
→00
 TEST2 NOW COMPLETE.0YOU ARE IN COMMAND MODE
 CHCIW STOP
→FTN TEST3,Y
A053 MODIFICATIONS? ENTER Y OR N.
→N
B016 LP FOUND NO ERRORS.
→TEST3

 TEST3: PLEASE ENTER N IN FORMAT I2
→10
 TEST3:ITERATIONS=  10
 START TIME= 19:16:13.01
 END   TIME= 19:18:07.48
 RESPONSE TIME=   114.47

 TEST3: PLEASE ENTER N IN FORMAT I2
→02
 TEST3:ITERATIONS=   2
 START TIME= 19:18:25.23
 END   TIME= 19:18:34.12
 RESPONSE TIME=     8.89

 TEST3: PLEASE ENTER N IN FORMAT I2
→00
 TEST3 NOW COMPLETE.0YOU ARE IN COMMAND MODE
 CHCIW STOP
→FINAL
CZUHW001 AT 19:19:00: 29 USERS, 14 TTY∇S, 15 2741∇S, 0 1050∇S
CZUHW001 AT 19:19:31: 29 USERS, 14 TTY∇S, 15 2741∇S, 0 1050∇S
B007   CPU TIME 00:01:24.77   CONNECT TIME 00:16:35   TIME 19:19   DATE 04/07/70
```

APPENDIX E

LISTING OF ALLOCATION MODEL

```
SIMULA BEGIN
INTEGER INDEX,SEED,K,COUNT,STATE,STARTUP,MAXCUSTOMERS;
INTEGER SNAP,MAXJOBS,MAXREQUESTS,MUSERS,DENIED;
REAL SUMS,SUMSS,SUMC,SUMCC,SUMPS,SUMP,SUMPP;
REAL STATTIME,STATEINTEGRAL,MARK,TEMP,OUT;
REAL ESR,LO1AV,LO2AV,LOURESPONSE,MICHRESPONSE,MARK2,CPIDLE,CPREST;
REAL MARK3,IOIDLE,IOREST;
REAL O1AV,O2AV,O1,O2,MARK4,MARK5;
REAL ARRAY DATA(1:21,1:2),INT1(1:20),INT2(1:20);
INTEGER ARRAY TRANSITDIST(1:21),SERVICEDIST(1:21),STATEPROB(1:21);
INTEGER ARRAY INT3(1:20);
SET ARRAY CPQUEUE(1:2),IOQUEUE(1:2);
REAL NAPR,ESLAM,DELT1,T1,MPF,ESPF;
REAL LAM;
ELEMENT PROCESSOR,IOSYS,MAIN;
ELEMENT UPDATE,GENUSERS;
BOOLEAN INITIALIZE;


PROCEDURE RESET;
BEGIN
    INITIALIZE:=FALSE;
    COUNT:=0;
    MARK:=MARK2:=MARK3:=MARK4:=MARK5:=STATTIME:=TIME;
    SUMS:=SUMSS:=SUMC:=SUMCC:=SUMPS:=SUMP:=SUMPP:=0.0;
    LO1AV:=LO2AV:=0.0;
    STATEINTEGRAL:=CPIDLE:=O1AV:=O2AV:=IOIDLE:=0.0;
    FOR K:=1 STEP 1 UNTIL 21 DO
        BEGIN
        TRANSITDIST(K):=SERVICEDIST(K):=STATEPROB(K):=0;
        DATA(K,1):=DATA(K,2):=0.0;
        END;
END OF PROCEDURE RESET;
```

```
ACTIVITY USER(PRIORITY) ; INTEGER PRIORITY;
BEGIN
REAL ARRIVALTIME,SERVICETIME,LSUMR,EXITPROB,THINKTIME,CYCLE1;
INTEGER INTERACTIONS,NUMREQUESTS;
BOOLEAN PASS1;
THINKTIME:=NEGEXP(1.0/30.0,SEED);
EXITPROB:=IF UNIFORM(0.0001,1.0,SEED) LEQ .75 THEN .5
          ELSE 1.0/26.0;
NUSERS:=NUSERS + 1;
NUMREQUESTS:=UNIFORM(0.0,20.0,SEED) + 1.0;
 U1:SERVICETIME:=CYCLE1:=0.0;
    PASS1:=TRUE;
    ARRIVALTIME:=TIME;
    HISTO(STATEPROB,INT3,STATE,(TIME-MARK)*100.0);
    ACCUM(STATEINTEGRAL,MARK,STATE,1);
    ACCUM(Q1AV,MARK4,Q1,1);
    NARR:=NARR+1.0;
    IF IDLE(PROCESSOR) THEN ACTIVATE PROCESSOR AFTER CURRENT;
    WAIT(CPQUEUE(PRIORITY));
    TEMP:=TIME-ARRIVALTIME;
    LSUMR:=LSUMR+TEMP;
    SUMR:=SUMR+TEMP;
    SUMRR:=SUMRR+TEMP*TEMP;
    SUMS:=SUMS+SERVICETIME;
    SUMSS:=SUMSS+SERVICETIME*SERVICETIME;
    SUMC:=SUMC+CYCLE1;
    SUMCC:=SUMCC+CYCLE1*CYCLE1;
    SUMRS:=SUMRS+TEMP*SERVICETIME;
    HISTO(TRANSITDIST,INT1,TEMP,1);
    HISTO(SERVICEDIST,INT2,SERVICETIME,1);
    HISTO(STATEPROB,INT3,STATE,(TIME-MARK)*100.0);
    INDEX:=SERVICETIME*10.0+1.0;
    ACCUM(STATEINTEGRAL,MARK,STATE,-1);
    ACCUM(Q2AV,MARK5,Q2,-1);
    INDEX:=IF INDEX LEQ 21 THEN INDEX ELSE 21;
    DATA(INDEX,1):=DATA(INDEX,1)+TEMP;
    DATA(INDEX,2):=DATA(INDEX,2)+1.0;
    COUNT:=COUNT+1;
    INTERACTIONS:=INTERACTIONS + 1;
    IF INTERACTIONS LSS NUMREQUESTS THEN
         BEGIN
         HOLD(UNIFORM(0.0,2.0*THINKTIME,SEED));
         GO TO U1;
         END;
NUSERS:=NUSERS - 1;
IF (DENIED GEQ 1) AND (NUSERS LSS MAXCUSTOMERS)
   THEN REACTIVATE GENUSERS AFTER CURRENT;
END OF ACTIVITY USER;
```

```
ACTIVITY ESTIMATOR;
BEGIN
REAL U1,U2,L,ERNEW,LAMNEW,ESM1,ESM2;
E1:ESLAM:=(ESLAM*T1+NARR)/(T1+DELT1);
    ESM1:=(ESM1*T1+Q1AV-LQ1AV)/(T1 + DELT1);
    ESM2:=(ESM2*T1+Q2AV-LQ2AV)/(T1 + DELT1);
    ESR:= ESM1 + ESM2;
    ESPF:=(ESPF*T1+NPF)/(T1+DELT1);
    IF (ESR LSS LOWRESPONSE) AND (NUSERS GEQ MAXCUSTOMERS) THEN
        BEGIN
         U1:=ESPF/ESM1 + ESPF;
         U2:=ESPF/ESM2 + ESPF;
         LAMNEW:= (NUSERS + 1)*ESPF/NUSERS;
         ERNEW:=LAMNEW*(1.0/(U1-LAMNEW) +1.0/(U2-LAMNEW));
         IF (ERNEW LSS HIGHRESPONSE) AND (DENIED GEQ 1) THEN
         BEGIN
             DENIED:=DENIED-1;
             MAXCUSTOMERS:=MAXCUSTOMERS + 1;
             ACTIVATE NEW USER(1) AFTER CURRENT;
         END;
    END
    ELSE IF (ESR GEQ HIGHRESPONSE) AND (MAXCUSTOMERS GEQ NUSERS)
          THEN MAXCUSTOMERS :=MAXCUSTOMERS-1;
    WRITE(NUSERS,MAXCUSTOMERS,ESR,ESLAM,ESPF,U1,U2);
    NARR:=NPF:=0.0;
    LQ1AV:=Q1AV;
    LQ2AV:=Q2AV;
    REACTIVATE CURRENT AT TIME + DELT1 PRIOR;
    GO TO E1;
END OF ACTIVITY ESTIMATOR;


ACTIVITY GENERATOR;
BEGIN
 G1:HOLD(NEGEXP(LAM,SEED));
    IF NUSERS GEQ MAXCUSTOMERS THEN
    BEGIN
         DENIED:=DENIED+1;
         GO TO G1;
    END;
    ACTIVATE NEW USER(1) AFTER CURRENT;
    GO TO G1;
END OF ACTIVITY GENERATOR;
```

```
ACTIVITY COMPUTER;
BEGIN
REAL OVERHEAD,QUANTA;
 C1:EXTRACT FIRST(CPQUEUE(1)) WHEN USER DO
          BEGIN
      C2:OVERHEAD:=.05 + STATE*.003;
          IF PASS1 THEN
               BEGIN
               CYCLE1:=TIME-ARRIVALTIME+OVERHEAD;
               PASS1:=FALSE;
               END;
          QUANTA:=UNIFORM(0.00001,.1,SEED);
          SERVICETIME:=SERVICETIME+QUANTA;
          HOLD(OVERHEAD+QUANTA);
          INCLUDE(USER,IOQUEUE(PRIORITY));
          ACCUM(Q1AV,MARK4,Q1,-1);
          ACCUM(Q2AV,MARK5,Q2,1);
          IF IDLE(IOSYS) THEN ACTIVATE IOSYS AFTER CURRENT;
          END
     OTHERWISE BEGIN
               ACCUM(CPIDLE,MARK2,CPREST,1.0);
              PASSIVATE;
               ACCUM(CPIDLE,MARK2,CPREST,-1.0);
          END;
     GO TO C1;
END OF ACTIVITY COMPUTER;


ACTIVITY IOPROCESSOR;
BEGIN
REAL IOSERVICE;
IO1: EXTRACT FIRST(IOQUEUE(1)) WHEN USER DO
          BEGIN
          NPF:=NPF+1;
          IOSERVICE:=UNIFORM(0.0,.2,SEED);
          HOLD(IOSERVICE);
          TEMP:=UNIFORM(0.0,1.0,SEED);
          IF TEMP LEQ EXITPROB THEN ACTIVATE USER AFTER CURRENT
               ELSE BEGIN
                    INCLUDE(USER,CPQUEUE(PRIORITY));
                    ACCUM(Q2AV,MARK5,Q2,-1);
                    ACCUM(Q1AV,MARK4,Q1,1);
                    IF IDLE(PROCESSOR) THEN
                         ACTIVATE PROCESSOR AFTER CURRENT;
                    END;
          END
     OTHERWISE BEGIN
          ACCUM(IOIDLE,MARK3,IOREST,1.0);
          PASSIVATE;
          ACCUM(IOIDLE,MARK3,IOREST,-1.0);
          END;
GO TO IO1;
END OF ACTIVITY IOPROCESSOR;
```

```
READ (SEED,STARTUP,MAXREQUESTS);
WRITE('  SEED',SEED,'  STARTUP',STARTUP,'MAXREQUESTS',MAXREQUESTS);
LAM:=1.0/10.0;
LOWRESPONSE:=3.25;
HIGHRESPONSE:=3.5;
MAXCUSTOMERS:= 30;
INITIALIZE:=TRUE;
T1:=50.0;
DELT1:=25.0;
FOR K:=1 STEP 1 UNTIL 20 DO
    BEGIN
    INT1(K):=1.5*K;
    INT2(K):=.1*K;
    INT3(K):=K-1;
    END;
PROCESSOR := NEW COMPUTER;
IOSYS:=NEW IOPROCESSOR;
UPDATE:=NEW ESTIMATOR;
GENUSERS:=NEW GENERATOR;
ACTIVATE UPDATE AT TIME +DELT1 + T1;
ACTIVATE GENUSERS AT TIME +100;
MAIN:=CURRENT;
FOR K:=1 STEP 1 UNTIL 20 DO
    ACTIVATE NEW USER(1) AFTER CURRENT;
FOR SNAP:= 1 STEP 1 UNTIL 16 DO
BEGIN
HOLD(400.0);
ACCUM(STATEINTEGRAL,MARK,STATE,0);
ACCUM(CPIDLE,MARK2,CPREST,0.0);
ACCUM(IOIDLE,MARK3,IOREST,0.0);
WRITE(' TIME AT RUN COMPLETION IS',TIME,' NUMBER OF TASKS PROCESSED IS',
    COUNT);
WRITE(' CURRENT STATE IS',STATE,'AVERAGE STATE IS',
    STATEINTEGRAL/(TIME-STATTIME));
TEMP:=SUMS/COUNT;
OUT:=SQRT(SUMSS/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF SERVICE TIMES ARE',TEMP,OUT);
TEMP:=(COUNT*SUMRS-SUMR*SUMS)/(COUNT*SUMSS-SUMS*SUMS);
OUT:=(SUMR-TEMP*SUMS)/COUNT;
WRITE(' REGRESSION ESTIMATE OF SLOPE AND INTERCEPT ARE',TEMP,OUT);
TEMP:=SUMC/COUNT;
OUT:=SQRT(SUMCC/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF FIRST WAITING TIMES ARE',TEMP,OUT);
TEMP:=SUMR/COUNT;
OUT:=SQRT(SUMRR/COUNT-TEMP*TEMP);
WRITE(' SAMPLE AVERAGE AND SD OF RESPONSE TIMES ARE',TEMP,OUT);
WRITE(' PROBABILITY PROCESSOR IS IDLE IS',CPIDLE/(TIME-STATTIME));
WRITE(' PROBABILITY IOSYSTEM IS IDLE IS',IOIDLE/(TIME-STATTIME));
WRITE(' AVERAGE NUMBER IN SYSTEM 1 AND 2 ARE',Q1AV/(TIME-STATTIME),
    Q2AV/(TIME-STATTIME));
WRITE(' THE TABLE DISPLAYS RESPONSE AS A FUNCTION OF SERVICE');
WRITE(' INTERVAL','AVERAGE R','NUMBER OF POINTS');
FOR K:=1 STEP 1 UNTIL 21 DO
    BEGIN
    TEMP:=IF DATA(K,2) GEQ 1.0 THEN DATA(K,1)/DATA(K,2) ELSE 0.0;
    OUT:=K/10;
    WRITE(OUT,TEMP,DATA(K,2));
    END;
RESET;
END;
END OF SIMULA BLOCK;
```

BIBLIOGRAPHY

Adiri, I., and B. Avi-Itzhak (1969), "A Time-Sharing Queue", Management Science, Vol. 15, No. 11, pp. 639-657.

Adiri, I. and B. Avi-Itzhak (1969b), "A Time-Sharing Queue with a Finite Number of Customers", Journal of the Association for Computing Machinery, Vol. 16, No. 2, pp. 315-323.

Avi-Itzhak, B. and P. Naor (1963), "Some Queueing Problems with the Service Station Subject to Breakdown", Operations Research, Vol. 11, No. 3, pp. 303-320.

Bell et al. (1971), C.mmp: The CMU Multiminiprocessor Computer, Requirements and Overview of the Initial Design, ARPA Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa.

Bryan, G. E. (1967), JOSS: 20,000 Hours at the Console - A Statistical Summary, Memorandum RM-5359-PR, The RAND Corporation.

Buzen, J. (1971), Queueing Network Models of Multiprogramming, Ph.D. Dissertation, Division of Engineering and Applied Physics, Harvard University, Cambridge, Mass.

Chang, W. (1966), "A Queueing Model for a Simple Case of Time-Sharing", IBM Systems Journal, Vol. 5, No. 2, pp. 115-125.

Coffman, E. G., and B. Krishnamoorthi (1964), Preliminary Analyses of Time-Shared Computer Operation, DOC. SP-1719, System Development Corporation, Santa Monica, California.

Coffman, E. G. (1966), Stochastic Models of Multiple and Time-Shared Computer Operations, Ph.D. Thesis, University of California, Los Angeles, California.

Coffman, E. G. and R. C. Wood (1966), "Interarrival Statistics for Time Sharing Systems", Communications of the ACM, Vol. 9, No. 7, pp. 500-503.

Coffman, E. G. and L. Kleinrock (1968), "Feedback Queueing Models for Time-Shared Systems", Journal of the Association for Computing Machinery, Vol. 15, No. 4, pp. 549-576.

Coffman, E. G. and R. R. Muntz (1969), "Models of Pure Time Sharing Disciplines for Resource Allocation", Proceedings of 24th National Conference of ACM, ACM Publication P-69, pp. 217-228.

Conway, C., W. Maxwell, L. Miller (1967), Theory of Scheduling, Addison-Wesley, Reading, Mass.

Corbato, F. (1968), "A Paging Experiment with the MULTICS System", MIT Project MAC, Document MAC-M-384.

Courtois, P. J. (1971), On the Near-Complete-Decomposability of Networks of Queues and of Stochastic Models of Multiprogramming Computing Systems, Report of Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa.

Dahl, O. and Nygaard (1966), "Simula - An Algol-Based Simulation Language", Communications of the ACM, Vol. 9, No. 9, pp. 671-678.

Deniston, W. (1969), "SIPE: A TSS/360 Software Measurement Technique", Proceedings of 24th National Conference of Association for Computing Machinery, ACM Publication P-69, New York, pp. 229-245.

Denning, P. (1968), "The Working Set Model for Program Behavior", Communications of the ACM, Vol. 11, No. 5, pp. 323-333.

Denning, P. J. (1970), "Virtual Memory", Computing Surveys, Vol. 2, No. 3, pp. 153-189.

Estrin, G. and L. Kleinrock (1967), "Measures, Models and Measurements for Time-Shared Computer Utilities", Proceedings of 22nd National Conference, Association for Computing Machinery, Thompson Book Co., Washington, D.C., pp. 85-96.

Feller, W. (1957), An Introduction to Probability Theory and its Applications, Vol. I, John Wiley and Sons, Inc., New York.

Gaver, D. P. (1969), "Statistical Methods for Improving Simulation Efficiency", Third Conference of Applications of Simulation, pp. 38-46.

Gordon, W. J. and C. F. Newell (1967), "Closed Queueing Systems with Exponential Servers", Operations Research, Vol. 15, No. 2, pp. 254-265.

Greenberger, M. (1966), "The Priority Problem and Computer Time-Sharing", Management Science, Vol. 12, No. 11, pp. 888-906.

IBM (1968), Time-Sharing System - Concepts and Facilities, Form c28-2003.

Jackson, J. R. (1963), "Jobshop-Like Queueing Systems", Management Science, Vol. 10, No. 1, pp. 132-142.

Kleinrock, L. (1964), "Analysis of a Time-Shared Processor", Naval Research Logistics Quarterly, Vol. 11, No. 1, pp. 59-73.

Kleinrock, L. (1967), "Time-Shared Systems: A Theoretical Treatment", Journal of the Association for Computing Machinery, Vol. 14, No. 2, pp. 242-261.

Krishnamoorthis, B. and R. C. Wood (1966), "Time-Shared Computer Operations with Both Interarrival and Service Times Exponential", Journal of the Association for Computing Machinery, Vol. 13, No. 3, pp. 317-338.

Little, J. (1961), "A Proof of the Queueing Formula $L=\lambda W$", Operations Research, Vol. 9, No. , pp. 383-387.

Little, John D. C. (1970), "Models and Managers: The Concept of a Decision Calculus", Management Science, Vol. 16, No. 8, pp. 466.

McCredie, J. W. (1967), A User Oriented Approach to the Design of Scheduling Algorithms for Time-Shared Computer Systems, Working Paper, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pa.

McCredie, J. W. (1970), "The Structure of Discrete Event Simulation Languages", Summer Simulation Conference Proceedings, pp. 88-98.

McCredie, J. W. and S. Schlesinger (1970), "A Modular Simulation of TSS/360", Applications of Simulation, pp. 201-206.

McKinney, J. M. (1969), "A Survey of Analytical Time-Sharing Models", Computing Surveys, Vol. 1., No. 2.

Madnick, S. W. (1968), "Multi-processor Software Lockout", Proceedings-1968 ACM National Conference, pp. 19-24.

Mills, R. G. (1971), "Appendix", Workshop on System Performance Evaluation, ACM, pp. 319-320.

Mood, A. M. and F. A. Graybill (1963), Introduction to the Theory of Statistics, McGraw-Hill Book Company, Inc., New York.

Moore, C. G. (1971), Network Models for Large-Scale Time-Sharing Systems, Ph.D. Dissertation, University of Michigan, Ann Arbor, Michigan, Technical Report No. 71-1.

Morse, P. M. (1958), Queues, Inventories and Maintenance, John Wiley and Sons, Inc., New York.

Parzen, E. (1962), Stochastic Processes, Holden-Day, Inc., San Francisco, California.

Rasch, P. J. (1970), "A Queueing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems", Journal of the Association for Computing Machinery, Vol. 17, No. 1, pp. 131-145.

Saaty, T. L. (1961), Elements of Queueing Theory with Applications, McGraw-Hill Book Company, Inc., New York.

Saaty, T. (1966), "Seven More Years of Queues, A Lament and a Bibliography", Naval Research Logistics Quarterly, Vol. 13, No. 4.

Schrage, L. E. (1966), Some Queueing Models for a Time-Shared Facility, Ph.D. Thesis, Cornell University, Ithaca, New York.

Schrage, L. (1969), The Modeling of Man-Machine Interactive Systems, University of Chicago Working Paper.

Scherr, A. L. (1967), An Analysis of Time-Shared Computer Systems, M.I.T. Research Monograph No. 36, The M.I.T. Press, Cambridge, Mass.

Simon, H. A., A. Ando (1961), "Aggregation of Variables in Dynamic Systems", Econometrica, Vol. 20, No. 2.

Takacs, L. (1963), "Single Server Queue with Feedback", The Bell Systems Technical Journal, Vol. LXII, No. 2, pp. 505-519.

Totschek, R. A. (1965), An Empirical Investigation into the Behavior of the SDC Time-Sharing System, SP-2191, System Development Corporation.

Univac (1970), SIMULA-Programmers Reference, UP-7556, Rev. 1, Sperry Rand Corporation.

Van de Goor, A. (1970), Design and Behavior of TSS/8: A PDP-8 Based Time-Sharing System, Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, Pa.

Wilkes, M. V. (1968), Time-Sharing Computer Systems, American Elsevier, New York.

Wilkes, M. V. (1971), "Automatic Load Adjustment in Time-Sharing Systems", Workshop on System Performance Evaluation, ACM, pp. 308-318.

Wulf, W. A. (1969), "Performance Monitors for Multi-Programmed Systems", Second Symposium on Operating Systems Principles, pp. 175-181.

Wulf, et al. (1971), HYDRA, A Kernel Operating System for C.mmp, External Specifications, Department of Computer Science Report, Carnegie-Mellon University, Pittsburgh, Pa.